

# GitLab Ultimate for IBM Cloud Paks ガイド

2021/06

オートメーション・テクニカル・セールス

# 目次

## 1. GitLab Ultimate for IBM Cloud Paks 概要

- GitLab Ultimate for IBM Cloud Paks とは
- GitLab の提供 / 導入形態
- GitLab Ultimate for IBM Cloud Paks の機能 / 無償版との違い

## 2. インストール

- 構成要素と構成パターン
- 導入方法パターン

### 2.1. Operator による GitLab のインストール

### 2.2. Operator による GitLab Runner のインストールと設定

### 2.3. ライセンスの設定

## 3. GitLab CI/CD の利用

### 3.1. GitLab CI/CD とは

### 3.2. GitLab CI/CD の用語

### 3.3. GitLab CI/CD の利用ステップ

## 4. Tips

# 1. GitLab Ultimate for IBM Cloud Paks 概要

# GitLab Ultimate for IBM Cloud Paks とは

## ■ GitLab とは? (<https://www.gitlab.jp>)

- ソフトウェア開発に必要な機能を統合した製品
  - 課題管理
  - バージョン管理
  - コードレビュー
  - CI/CD
  - モニタリング

## ■ GitLab Ultimate 固有の機能は? (<https://www.gitlab.jp/pricing/ultimate/>)

- セキュリティ
- コンプライアンス
- プロジェクトとグループについての洞察
- ポートフォリオ管理

## ■ GitLab Ultimate for IBM Cloud Paks とは?

- IBM 版の GitLab Ultimate
  - GitLab 社から IBM へ OEM 提供
  - L1 サポートは IBM が実施

# GitLab の導入 / 提供形態

- GitLab.com SaaS
  
- GitLab self-managed
  - Linux パッケージ (dev/rpm)
    - Omibus GitLab: 前提の PostgreSQL, Redis, Sidekiq を含む
  - Helm チャート
  - Docker
  - ソースからビルド
  - GitLab Environment Toolkit (GET)
    - GCP, AWS, Azure (予定)
      - Terraform で Provisioning, Ansible で Configuration
  
  - Operator



今回の検証

# エディションごとの機能 (1)

## ■ 最新の情報はこちらを参照下さい

– <https://about.gitlab.com/pricing/self-managed/feature-comparison/>

	Free	Premium	Ultimate
Built-in CI/CD	○	○	○
Project Issue Board	○	○	○
Group Issue Board	○	○	○
Multiple Project Issue Boards	○	○	○
Time Tracking	○	○	○
Git LFS 2.0 support	○	○	○
Wiki based project documentation	○	○	○
WYSIWYG Editing in Wiki	○	○	○
Design Management	○	○	○
GitLab-Figma Plugin	○	○	○
Project Level Value Stream Analytics	○	○	○
Preview your changes with Review Apps	○	○	○
Environments Auto-stop	○	○	○
Snippets	○	○	○
Publish static websites for free with GitLab Pages	○	○	○
Keep track of releases using GitLab Releases	○	○	○
Group-level release analytics	○	○	○
Secret Detection	○	○	○
Static Site Editor	○	○	○
Scoped Labels		○	○
Issue Weights		○	○
Iterations		○	○
Multiple Issue Assignees		○	○
Issue Dependencies		○	○
Epic Boards		○	○
Bulk Edit Epics		○	○
Burnup Charts		○	○
Burndown Charts		○	○
Custom Text in Emails		○	○
Track Description Changes		○	○
Multiple Group Issue Boards		○	○
Total Issue Weight per Issue Board List		○	○
Issue Board Assignee Lists		○	○
Issue Board Milestone Lists		○	○
Issue Board Configuration		○	○
Single level Epics		○	○
Confidential Epics		○	○
Reorder Issues in Epic Tree		○	○
Epic Fixed Dates		○	○

	Free	Premium	Ultimate
Epic Dynamic Dates		○	○
Promote Issue to Epic		○	○
Roadmaps		○	○
Issue Analytics		○	○
Required Merge Request Approvals		○	○
Multiple approvers in code review		○	○
Approval rules for code review		○	○
Repository pull mirroring		○	○
Push rules		○	○
Block secret file push		○	○
Reject unsigned commits		○	○
Verified Committer		○	○
Restrict push and merge access to certain users		○	○
Instance file templates		○	○
Group file templates		○	○
Code Owners		○	○
Scalable fault-tolerant Git storage with Gitly Cluster		○	○
Variable replication factor		○	○
Group-level Wiki		○	○
Group Level Value Stream Analytics		○	○
Priority Support		○	○
24/7 uptime support		○	○
Next business day Support		○	○
Multiple LDAP / AD server support		○	○
Advanced Search		○	○
Create and remove admins based on an LDAP group		○	○
Kerberos user authentication		○	○
Group webhooks		○	○
Fault-tolerant PostgreSQL		○	○
Email all users of a project, group, or entire server		○	○
Limit project size at a global, group, and project level		○	○
Omnibus package supports log forwarding		○	○
Lock project membership to group		○	○
LDAP group sync		○	○
LDAP group sync filters		○	○
Live upgrade assistance		○	○
Audit Events		○	○
Auditor users		○	○
Disaster Recovery		○	○
CI/CD Pipelines Dashboard		○	○

# エディションごとの機能 (2)

	Free	Premium	Ultimate
Container Registry geographic replication		○	○
Code Quality Reports		○	○
Multi-project pipeline graphs		○	○
GitLab Kubernetes Agent		○	○
Globally distributed cloning with GitLab Geo		○	○
Support for Scaled Architectures		○	○
Built-in and custom project templates		○	○
Performance Testing		○	○
Group Code Coverage Data		○	○
Contribution Analytics		○	○
CI/CD for external repo		○	○
CI/CD for GitHub		○	○
SAML SSO for Groups		○	○
Supports geolocation-aware DNS		○	○
Restrict access by IP address		○	○
Protected Environments		○	○
Comment directly on Review Apps		○	○
Associate Feature Flags with the issue(s) that is related to them		○	○
Smart card support		○	○
Pipelines for Merged Results		○	○
Merge Trains		○	○
Run pipelines in the parent project for MRs from forks		○	○
Environments Dashboard		○	○
Cross-project jobs with artifact dependencies		○	○
Productivity Analytics		○	○
Code Review Analytics		○	○
Cluster Environments Global View		○	○
Merge Request Dependencies		○	○
Compliance Frameworks		○	○
Export a user access report		○	○
Manage access to protected environments from the API		○	○
View Jira issues in GitLab		○	○
Service Level Agreement countdown timer		○	○
On-call Schedule Management		○	○
Maintenance mode		○	○
Static Application Security Testing		○	○
Multi-level Epics			○
Issue and Epic Health Reporting			○
Portfolio-level Roadmaps			○
Requirements Management			○

	Free	Premium	Ultimate
Satisfy Requirements from CI/CD pipelines			○
Import & Export Requirements			○
Quality Management			○
Create test cases from within GitLab			○
Portfolio Management			○
Free Guest users			○
Compliance framework default pipelines			○
Configuration UI			○
Custom Rulesets for SAST			○
Custom Rulesets for Secret Detection			○
Vulnerability Database			○
Project Dependency List			○
Dependency Scanning			○
Dynamic Application Security Testing			○
Dynamic Application Security Testing support for REST API scans			○
Automated solutions for Dependency Scanning vulnerabilities			○
Vulnerability Management			○
Standalone Vulnerability Objects			○
Security Dashboards			○
Policy Management for Container Network Policies			○
Security Alert Dashboard for Container Network Policy Alerts			○
Container Scanning			○
Automated solutions for Container Scanning vulnerabilities			○
Security Approvals			○
License Compliance			○
Deployment Frequency metrics			○
Track DORA-4 lead time for changes metric			○
Track DORA-4 Deployment frequency charts			○
Insights			○
View alerts on the Environments page			○
View deployment status on the Environments page			○
Credentials Management			○
Compliance Dashboard			○
Status Page			○
Auto Rollback in case of failure			○
Coverage-guided Fuzz Testing			○
API Fuzz Testing			○
On-demand Dynamic Application Security Testing			○
Site and Scanner profiles for On-demand DAST scans			○

構成要素と構成パターン

## 2. インストール

# GitLab の構成要素 (1)

## ■ GitLab (サーバー)

– (1. GitLab Ultimate for IBM Cloud Paks 概要 - GitLab の導入 / 提供形態 を参照)

## ■ GitLab Runner

– CI/CD パイプラインを実行するために必要

– 以下のアーキテクチャーをサポート

- x86
- AMD64
- ARM64
- ARM
- s390x

– 以下のプラットフォームをサポート

- GNU/Linux
- macOS
- Windows
- FreeBSD
- Docker
- Kubernetes / GitLab Kubernetes Agent
- OpenShift

## GitLab の構成要素 (2)

- CI/CD 成果物を OpenShift にデプロイするために、GitLab サーバーおよび GitLab Runner を OpenShift にインストールすることは必須ではない。
  - GitLab サーバーは本質的には以下の集合体
    - プロジェクト・リポジトリ
    - ユーザー・インターフェース
    - ユーザーインターフェースからリポジトリを操作するためのコンポーネント
  - GitLab Runner は CI/CD の実行エンジン
  - よって、必ずしもデプロイ先で稼働する必要はない。
  
- GitLab Runner をコンテナ環境で稼働させる (= Docker in Docker) には考慮点が存在する。
  - 特権ユーザー (= root) を必要とするので、セキュリティ上の課題がある。
  - 一般にパフォーマンスに難があるとされる。
  - 今回の検証では kaniko を使用することでこれらの考慮点を克服している。
    - 「3.3. GitLab CI/CD利用ステップ - 3. .gitlab-ci.yml ファイルの作成」 - 「.gitlab-ci.yml ファイルの記述例」 - 「コンテナイメージのビルドとレジストリーへの格納」を参照下さい。
    - kaniko について詳しくは [https://docs.gitlab.com/ee/ci/docker/using\\_kaniko.html](https://docs.gitlab.com/ee/ci/docker/using_kaniko.html) を参照下さい。

# 導入方法パターン

## ■ 導入方法のパターン

- (1. GitLab Ultimate for IBM Cloud Paks 概要 - GitLab の導入 / 提供形態 の再掲)
  - Linux パッケージ (Omnibus GitLab)
  - Helm チャート
  - Docker
  - ソースからビルド
  - GitLab Environment Toolkit
  - Operator

## ■ 実行環境を OpenShift とする場合

- Helm チャートは OpenShift を未サポート
  - GitLab Docs - Installing GitLab on OKD (OpenShift Origin)
    - <https://docs.gitlab.com/charts/installation/cloud/openshift.html>
    - "Currently GitLab does not target or provide support for OpenShift Installations."
- よって、事実上 Operator の一択となる。
  - 今回の検証も Operator を使用。
  - ただし、本ガイド作成時点で Beta として提供。
  - GitLab Docs - GitLab Operator
    - <https://docs.gitlab.com/charts/installation/operator.html>
    - " A GitLab Operator is now available in Beta. "

## ■ 本ガイド作成時点での GitLab self-managed の推奨実行環境

- Linux パッケージ (オールインワンパッケージ / Omnibus GitLab)
  - <https://www.gitlab.jp/install/> を参照下さい。

2.1. Operator による GitLab のインストール

## 2. インストール

# インストールの流れ

- 以下の手順でインストールを行う

**1**

OperatorによるGitLabのインストール

**2**

OperatorによるGitLab Runnerのインストール

**3**

ライセンスの設定

# インストールの流れ

- 以下の手順でインストールを行う

**1**

OperatorによるGitLabのインストール

**2**

OperatorによるGitLab Runnerのインストール

**3**

ライセンスの設定

## 2.1. OperatorによるGitLabのインストール

- 通常、OpenShiftでOperatorによるインストールを行う場合、OpenShiftのWebコンソールを利用する
- 本ガイド作成時点(2021年6月末)では、GitLab Operatorが正式リリース前でOpenShiftのWebコンソールからは導入できないため、下記リンク先を参照したGitLab Operatorのリポジトリの自動インストール手順を説明する

<https://gitlab.com/gitlab-org/gl-openshift/gitlab-operator/-/blob/master/doc/installation.md>

– 補足：ガイド作成時点で参照したコミットは以下。

<https://gitlab.com/gitlab-org/gl-openshift/gitlab-operator/-/tree/c9d23b266e327f19d3454acce1664b1d607868c4>

- 今回のインストールで使用するOperator(とNamespace)は右図

Project: all projects ▾

### Installed Operators

Installed Operators are represented by Cluster Service Versions within this namespace. For more information, see the [Understanding Operators documentation](#). Or create an Operator and Cluster Service Version using the [Operator SDK](#).

Name ▾ Search by name... 🔍

Name ↑	Namespace ↑	Managed Namespaces ↑	Status	Last Updated	Provided APIs
 <b>cert-manager</b> 1.1.0 provided by Jetstack	NS openshift-operators	All Namespaces	✔ Succeeded Up to date	🕒 Apr 27, 9:15 am	CertManager
 <b>GitLab Runner</b> 1.0.0 provided by GitLab, Inc.	NS gitlab-system	NS gitlab-system	⚠ Cannot update Catalog source was removed.	🕒 Jun 29, 1:53 am	GitLab Runner
 <b>GitLab Runner</b> 1.1.0 provided by GitLab, Inc.	NS gitlab-system	NS gitlab-system	❌ Failed Upgrade available	🕒 Jul 2, 11:00 am	GitLab Runner
 <b>Nginx Ingress Operator</b> 0.2.0 provided by NGINX Inc.	NS nginx-temp	NS nginx-temp	✔ Succeeded 🔄 Upgrade available	🕒 May 21, 6:56 pm	NginxIngressController
 <b>Package Server</b> 0.16.1 provided by Red Hat	NS openshift-operator-lifecycle-manager	NS openshift-operator-lifecycle-manager	✔ Succeeded	🕒 Jul 18, 9:24 am	PackageManifest

# ガイド作成環境

## ■ Red Hat OpenShift on IBM Cloud クラスター

- インフラストラクチャー：クラシック
- 可用性：単一ゾーン
- ワーカー・プール：仮想 共用 RHEL 4vCPU 16GB
- ゾーンあたりのワーカー・ノード：3
- OpenShift Version 4.6.22
- 今回の検証では、20GiB x 3、50GiB x 1 の PV が自動的に作成された。

### – 導入SW

- GitLab Enterprise Edition 13.10.3-ee
- GitLab Runner 13.10

※GitLab画面については、以下の手順でGUIを日本語にした状態での手順を記載（一部例外あり）

– GitLab GUI の右上の自分のアイコン → Preferences → Localization → Language を Japanese 日本語にする

## ■ 端末

### – MacBook Air macOS Catalina

- git version 2.28.0
- GNU Make 3.81
- go version go1.16.3 darwin/amd64
- oc Client Version: 4.6.28

## 2.1. OperatorによるGitLabのインストール – 導入の流れ

導入の流れは以下の通り

1. 端末の準備
2. ドメイン名の準備 (default routeの確認)
3. NGINX Ingress の導入
4. Cert Manager の導入
5. OperatorとGitLabの導入
6. 接続確認

## 2.1. OperatorによるGitLabのインストール – 導入の流れ

導入の流れは以下の通り

1. **端末の準備**
2. ドメイン名の準備（default routeの確認）
3. NGINX Ingress の導入
4. Cert Manager の導入
5. OperatorとGitLabの導入
6. 接続確認

## 2.1. OperatorによるGitLabのインストール – 1.端末の準備

インストールを行う端末に以下のツールを導入しておく

- git cli

- 参考 : [Gitのインストール](#)

- oc cli

- 参考 : [OpenShift CLIのインストール](#)

- makeコマンド

- go バージョン1.12以降

- 参考 : [Go Download and install](#)

## 2.1. OperatorによるGitLabのインストール – 導入の流れ

導入の流れは以下の通り

1. 端末の準備
2. ドメイン名の準備 (default routeの確認)
3. NGINX Ingress の導入
4. Cert Manager の導入
5. OperatorとGitLabの導入
6. 接続確認

## 2.1. OperatorによるGitLabのインストール – 2.ドメイン名の準備 (default routeの確認)

1. OpenShiftコンソールにログインし、左側のメニューから「Networking」を展開し「Routes」をクリックする

The screenshot displays the OpenShift console interface. The left-hand navigation menu is expanded to show 'Networking', with 'Routes' selected and highlighted by a red box. The main content area shows the 'Overview' page for the 'Cluster' section. The 'Routes' page is visible, showing details for the cluster's API address, ID, provider (IBMCloud), and version (4.6.22). The 'Status' section indicates that both 'Cluster' and 'Operators' are in a healthy state. A warning message is displayed, stating that alerts are not configured to be sent to a notification system. The 'Cluster Utilization' section shows a graph for resource usage over time, with a dropdown menu set to '1 Hour'. The 'Activity' section shows a list of recent events, including 'Stopping container...', 'Created container...', 'Successfully pulled...', 'Started container...', 'Pulling image...', 'Add eth0 [172.30...', 'Successfully assign...', and 'Stopping container...'. The URL at the bottom of the browser window is `https://console-openshift-console.mycluster-tok02-h-889642-d85540f5ac7eb4665421e4a86h8d368-0000.in-tok.containers.appdomain.cloud/k8s/all-namespaces/routes`.

## 2.1. OperatorによるGitLabのインストール – 2.ドメイン名の準備 (default routeの確認)

2. 一覧の上部の入力欄に「router-default」と入力し、デフォルトのrouterを表示する。表示されたName欄の「router-default」をクリックする

The screenshot shows the Red Hat OpenShift Container Platform console interface. The left sidebar contains navigation options: Home, Overview, Projects, Search, Explore, Events, Operators, Workloads, Networking, Services, Routes, Ingresses, and Network Policies. The main content area displays the 'Routes' page for the 'all projects' namespace. A search filter is applied to the 'Name' column, showing 'router-default'. The resulting table lists one route:

Name ↑	Namespace ↓	Status	Location ↓	Service ↓
<b>RT</b> router-default	<b>NS</b> openshift-ingress	✓ Accepted	http://router-default.mycluster-tok02-b-889642-d85540d5ac7eb4665421ec4a86b8d368-0000.jp-tok.containers.appdomain.cloud/healthz	<b>S</b> router-internal-default

## 2.1. OperatorによるGitLabのインストール – 2.ドメイン名の準備 (default routeの確認)

- 表示されたrouter-defaultのRouter Canonical Hostname の値を控えておく
  - この値がGitLabにアクセスする際のURLのサブ・ドメインとなる

The screenshot shows the Red Hat OpenShift Container Platform console. The left sidebar contains navigation options: Home, Overview, Projects, Search, Explore, Events, Operators, Workloads, Networking (selected), Services, Routes (highlighted), Ingresses, and Network Policies. The main content area displays the details for a Route named 'router-default' in the 'openshift-ingress' namespace. The 'Router Canonical Hostname' field is highlighted with a red box and contains the value 'mycluster-tok02-b-889642-d85540d5ac7eb4665421ec4a86b8d368-0000.jp-tok.containers.appdomain.cloud'. A tooltip below it asks 'Do you need to set up custom DNS?'. Other fields include Name, Location, Namespace, Labels, Annotations, Service, Target Port, Created At, and Owner.

Field	Value
Name	router-default
Location	<a href="http://router-default.mycluster-tok02-b-889642-d85540d5ac7eb4665421ec4a86b8d368-0000.jp-tok.containers.appdomain.cloud/healthz">http://router-default.mycluster-tok02-b-889642-d85540d5ac7eb4665421ec4a86b8d368-0000.jp-tok.containers.appdomain.cloud/healthz</a>
Namespace	NS openshift-ingress
Labels	<code>ingresscontroller.operator.openshift.io/owning-ingresscontroller=default</code>
Annotations	2 Annotations
Service	S router-internal-default
Target Port	1936
Created At	Apr 13, 2:43 pm
Owner	
Status	Accepted
Host	router-default.mycluster-tok02-b-889642-d85540d5ac7eb4665421ec4a86b8d368-0000.jp-tok.containers.appdomain.cloud
Path	/healthz
Router Canonical Hostname	mycluster-tok02-b-889642-d85540d5ac7eb4665421ec4a86b8d368-0000.jp-tok.containers.appdomain.cloud

## 2.1. OperatorによるGitLabのインストール – 導入の流れ

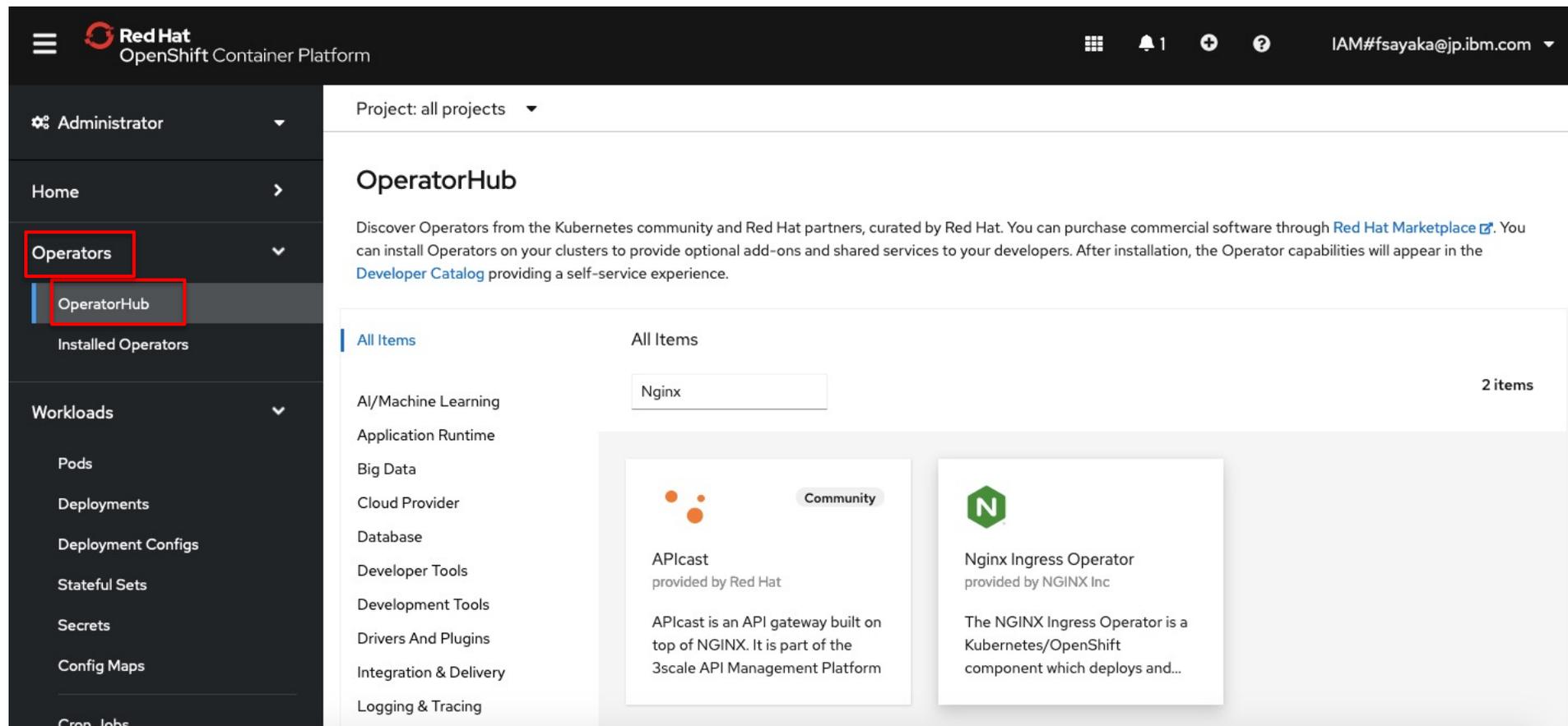
導入の流れは以下の通り

1. 端末の準備
2. ドメイン名の準備 (default routeの確認)
3. **NGINX Ingress の導入**
4. Cert Manager の導入
5. OperatorとGitLabの導入
6. 接続確認

## 2.1. OperatorによるGitLabのインストール –3.NGINX Ingressの導入

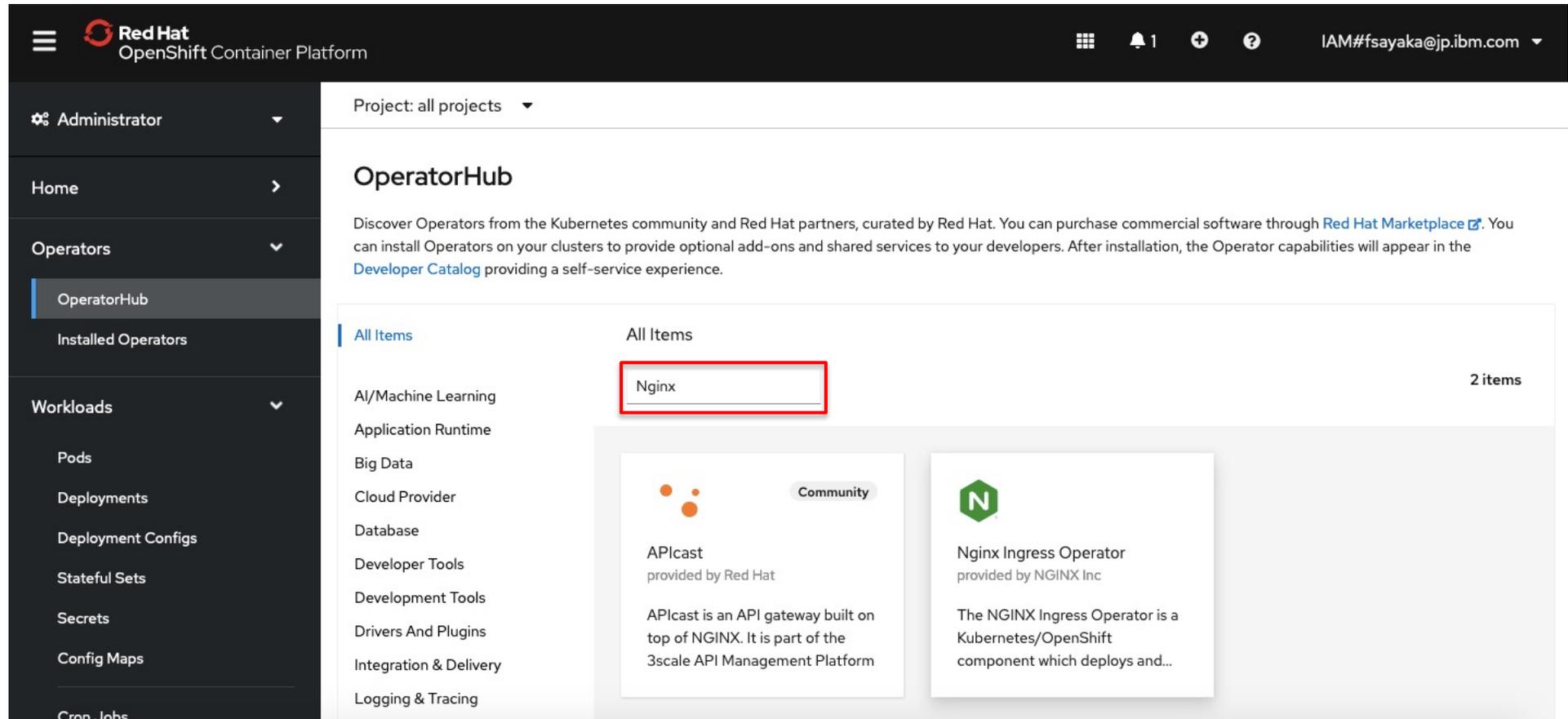
- [前提要件](#)に記載されているNGINX IngressをOperatorを使用して導入する

1. 管理者権限のあるユーザーでOpenShiftコンソールにログインし、左側のメニューから「Operators」を展開し「OperatorHub」をクリックする



## 2.1. OperatorによるGitLabのインストール –3.NGINX Ingressの導入

2.一覧の上部の入力欄に「Nginx」と入力しNginx IngressのOperatorを表示する



The screenshot shows the Red Hat OpenShift Container Platform interface. The top navigation bar includes the Red Hat logo, the text 'OpenShift Container Platform', and user information 'IAM#fsayaka@jp.ibm.com'. The left sidebar contains navigation options: Administrator, Home, Operators (selected), Installed Operators, Workloads, Pods, Deployments, Deployment Configs, Stateful Sets, Secrets, Config Maps, and Cron Jobs. The main content area is titled 'OperatorHub' and contains a search bar with the text 'Nginx' entered. Below the search bar, there are two operator cards: 'APIcast provided by Red Hat' and 'Nginx Ingress Operator provided by NGINX Inc'. The 'Nginx Ingress Operator' card is highlighted with a red box.

## 2.1. OperatorによるGitLabのインストール – 3. NGINX Ingressの導入

### 3. Nginx Ingress Operatorをクリックする

The screenshot displays the Red Hat OpenShift Container Platform OperatorHub interface. The top navigation bar includes the Red Hat logo, the text "Red Hat OpenShift Container Platform", and user information "IAM#fsayaka@jp.ibm.com". The left sidebar contains navigation options: Administrator, Home, Operators (selected), Installed Operators, Workloads, Pods, Deployments, Deployment Configs, Stateful Sets, Secrets, Config Maps, and Cron Jobs. The main content area shows the "OperatorHub" page with a search filter set to "Nginx" and "2 items" displayed. The "Nginx Ingress Operator" is highlighted with a red box. The interface also shows a list of categories on the left, including AI/Machine Learning, Application Runtime, Big Data, Cloud Provider, Database, Developer Tools, Development Tools, Drivers And Plugins, Integration & Delivery, and Logging & Tracing.

Project: all projects

### OperatorHub

Discover Operators from the Kubernetes community and Red Hat partners, curated by Red Hat. You can purchase commercial software through [Red Hat Marketplace](#). You can install Operators on your clusters to provide optional add-ons and shared services to your developers. After installation, the Operator capabilities will appear in the [Developer Catalog](#) providing a self-service experience.

All Items

2 items

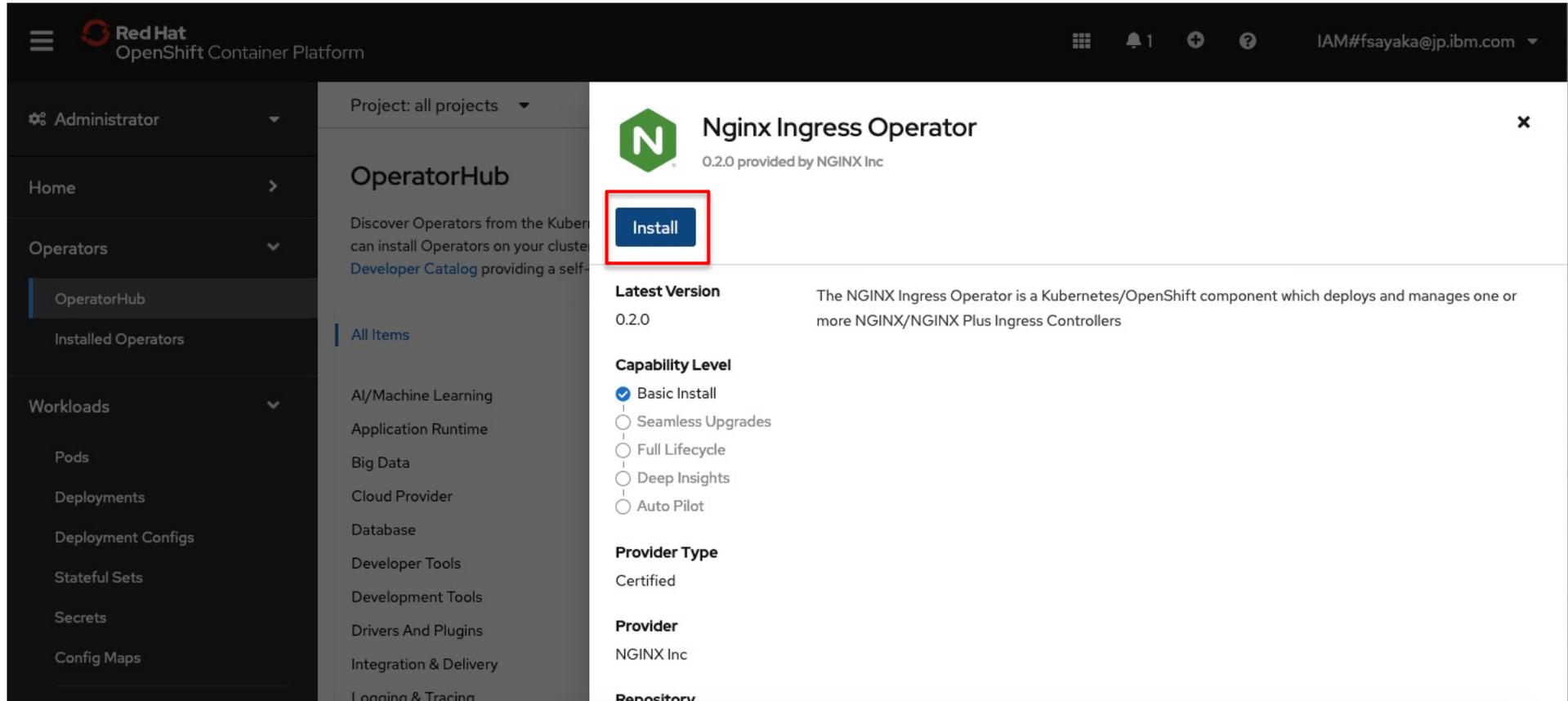
Community

**Ngix Ingress Operator**  
provided by NGINX Inc

The NGINX Ingress Operator is a Kubernetes/OpenShift component which deploys and...

## 2.1. OperatorによるGitLabのインストール –3.NGINX Ingressの導入

### 4. Nginx Ingress Operatorの画面で「Install」をクリックする



The screenshot displays the Red Hat OpenShift OperatorHub interface. The top navigation bar includes the Red Hat logo, the text "Red Hat OpenShift Container Platform", and user information "IAM#fsayaka@jp.ibm.com". The left sidebar shows navigation options: Administrator, Home, Operators (with OperatorHub selected), Installed Operators, and Workloads (with Pods, Deployments, Deployment Configs, Stateful Sets, Secrets, and Config Maps listed). The main content area is titled "OperatorHub" and shows "Project: all projects". The "All Items" section lists various categories like AI/Machine Learning, Application Runtime, Big Data, Cloud Provider, Database, Developer Tools, Development Tools, Drivers And Plugins, Integration & Delivery, and Logging & Tracing. The "Nginx Ingress Operator" card is prominently displayed, featuring the NGINX logo, the version "0.2.0 provided by NGINX Inc", and a blue "Install" button highlighted with a red border. Below the card, details are provided: "Latest Version" is 0.2.0, "Capability Level" includes "Basic Install" (selected), "Seamless Upgrades", "Full Lifecycle", "Deep Insights", and "Auto Pilot", "Provider Type" is "Certified", and "Provider" is "NGINX Inc".

## 2.1. OperatorによるGitLabのインストール – 3. NGINX Ingressの導入

### 5. 表示された画面で「Install」をクリックする

- 導入先のネームスペースや更新の方法（自動または手動）は必要であれば変更する

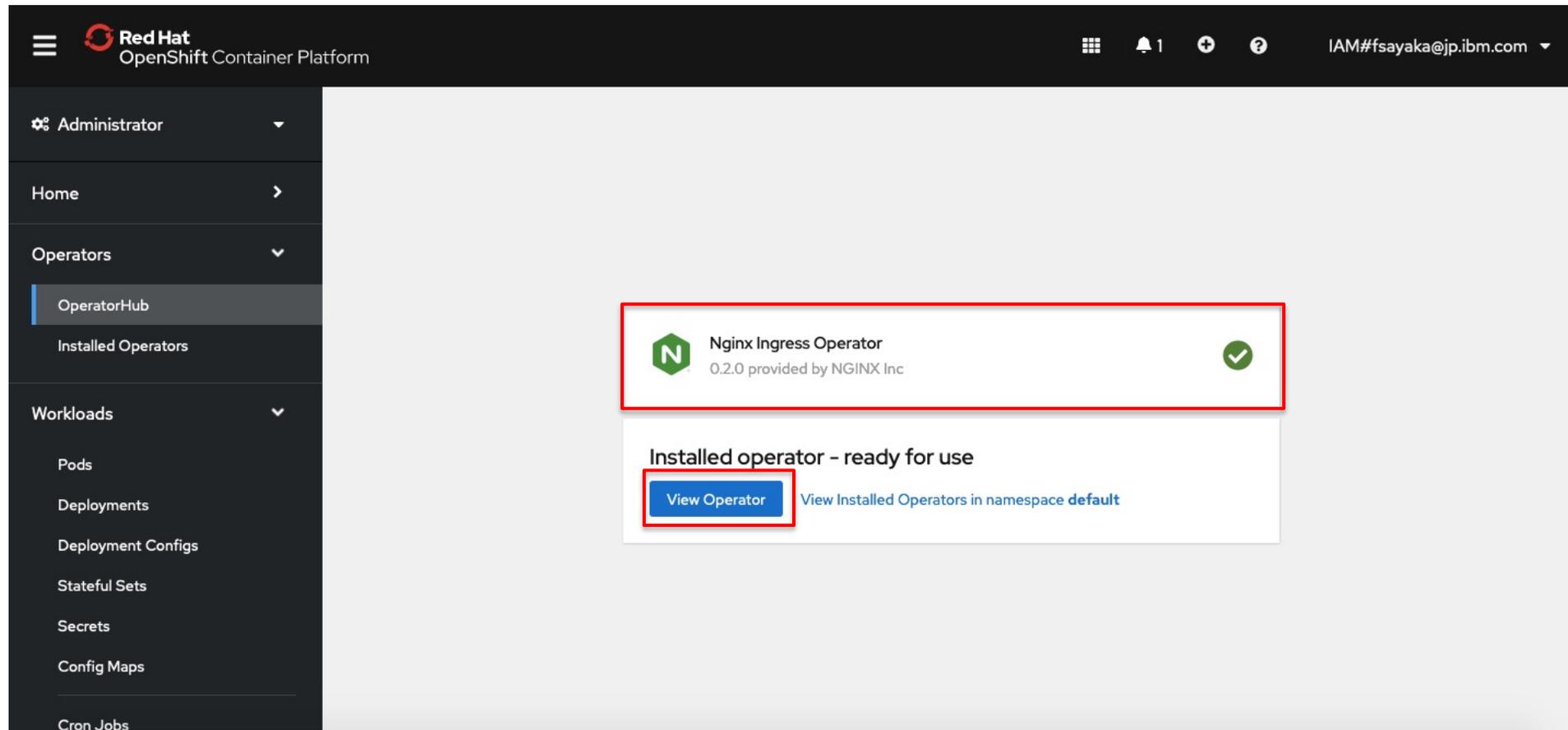
The screenshot shows the Red Hat OpenShift Container Platform OperatorHub interface. The left sidebar contains navigation options: Administrator, Home, Operators (selected), OperatorHub (highlighted), Installed Operators, Workloads (expanded), Pods, Deployments, Deployment Configs, Stateful Sets, Secrets, Config Maps, and Cron Jobs. The main content area displays the configuration for the Nginx Ingress Operator, provided by NGINX Inc. The configuration includes:

- Update Channel:** alpha (selected)
- Installation Mode:** A specific namespace on the cluster (selected). A note states: "This mode is not supported by this Operator. Operator will be available in a single namespace only."
- Installed Namespace:** default (selected in the dropdown menu)
- Approval Strategy:** Automatic (selected)

Under "Provided APIs", the NginxIngressController is listed with the note: "NginxIngressController is the Schema for the nginxingresscontrollers API". At the bottom, the "Install" button is highlighted with a red box, and a "Cancel" button is also visible.

## 2.1. OperatorによるGitLabのインストール – 3. NGINX Ingressの導入

6. 導入が完了すると以下のように緑色のチェックマークが表示される  
「View Operator」をクリックすると詳細を確認できる



## 2.1. OperatorによるGitLabのインストール – 3. NGINX Ingressの導入

### 7. 表示された画面でNginx Ingress Operatorの詳細を確認できる

The screenshot shows the Red Hat OpenShift Container Platform interface. The sidebar on the left contains navigation options: Administrator, Home, Operators, Workloads, and Cron Jobs. The main content area displays the details of the Nginx Ingress Operator. The operator's name is "Nginx Ingress Operator" with version "0.2.0 provided by NGINX Inc". There is an "Actions" dropdown menu. Below this, there are tabs for "Details", "YAML", "Subscription", "Events", and "NginxIngressController". The "Details" tab is active, showing "Provided APIs" with a "NIC NginxIngressController" and a "Create Instance" button. The "Description" section states: "The NGINX Ingress Operator is a Kubernetes/OpenShift component which deploys and manages one or more NGINX/NGINX Plus Ingress Controllers". On the right side, there are details for "Provider" (NGINX Inc), "Created At" (less than a minute ago), and "Links" (Documentation for NGINX Plus and Documentation for NGINX Open Source).

- 本ガイド作成時に参照したインストール手順ではNginx Ingress Operatorについては、導入までが前提要件となっていたため、インスタンスの作成は行なっていない。
- インストール実施時点の手順に従って実施のこと。

## 2.1. OperatorによるGitLabのインストール – 導入の流れ

導入の流れは以下の通り

1. 端末の準備
2. ドメイン名の準備（default routeの確認）
3. NGINX Ingress の導入
4. **Cert Manager の導入**
5. OperatorとGitLabの導入
6. 接続確認

## 2.1. OperatorによるGitLabのインストール – 4.Cert Manager の導入

- GitLabやレジストリーへのアクセス時に使用する証明書を作成するためのCert Manager operatorを導入する

1. OpenShiftコンソールにログインし、左側のメニューから「Operators」を展開し「OperatorHub」をクリックする

The screenshot shows the Red Hat OpenShift Container Platform interface. The top navigation bar includes the Red Hat logo, the text 'OpenShift Container Platform', and the user's identity 'IAM#fsayaka@jp.ibm.com'. The left sidebar contains a menu with 'Operators' expanded, and 'OperatorHub' highlighted. The main content area is titled 'OperatorHub' and contains a search bar with 'cert-manager' entered. Below the search bar, two search results are displayed, both for 'cert-manager provided by Jetstack'.

Category	Item Name	Provider	Description
AI/Machine Learning	cert-manager	Jetstack	x509 certificate management for Kubernetes
Application Runtime	cert-manager	Jetstack	x509 certificate management for Kubernetes

## 2.1. OperatorによるGitLabのインストール – 4.Cert Manager の導入

2.一覧の上部の入力欄に「cert-manager」と入力しcert-manager Operatorを表示する

The screenshot shows the Red Hat OpenShift Container Platform interface. The top navigation bar includes the Red Hat logo, the text "OpenShift Container Platform", and the user email "IAM#fsayaka@jp.ibm.com". The left sidebar contains a navigation menu with items like "Administrator", "Home", "Operators", "OperatorHub", "Installed Operators", "Workloads", "Pods", "Deployments", "Deployment Configs", "Stateful Sets", "Secrets", "Config Maps", and "Cron Jobs". The "OperatorHub" section is active, showing a search bar with "cert-manager" entered and highlighted. Below the search bar, there are two operator cards for "cert-manager" provided by Jetstack. The first card is the default view, and the second card has a "Marketplace" badge. The text "2 items" is visible on the right side of the search results.

## 2.1. OperatorによるGitLabのインストール – 4.Cert Manager の導入

3.一覧の上部の入力欄に「cert-manager」（Marketplaceの表示がない方）をクリックする

The screenshot shows the Red Hat OpenShift Container Platform OperatorHub interface. The left sidebar contains navigation options: Administrator, Home, Operators (selected), Installed Operators, Workloads, Pods, Deployments, Deployment Configs, Stateful Sets, Secrets, Config Maps, and Cron Jobs. The main content area displays the OperatorHub page with a search bar containing 'cert-manager' and two search results. The first result, 'cert-manager provided by Jetstack', is highlighted with a red box. The second result, 'cert-manager provided by Jetstack', is marked as 'Marketplace'.

Project: all projects

### OperatorHub

Discover Operators from the Kubernetes community and Red Hat partners, curated by Red Hat. You can purchase commercial software through [Red Hat Marketplace](#). You can install Operators on your clusters to provide optional add-ons and shared services to your developers. After installation, the Operator capabilities will appear in the [Developer Catalog](#) providing a self-service experience.

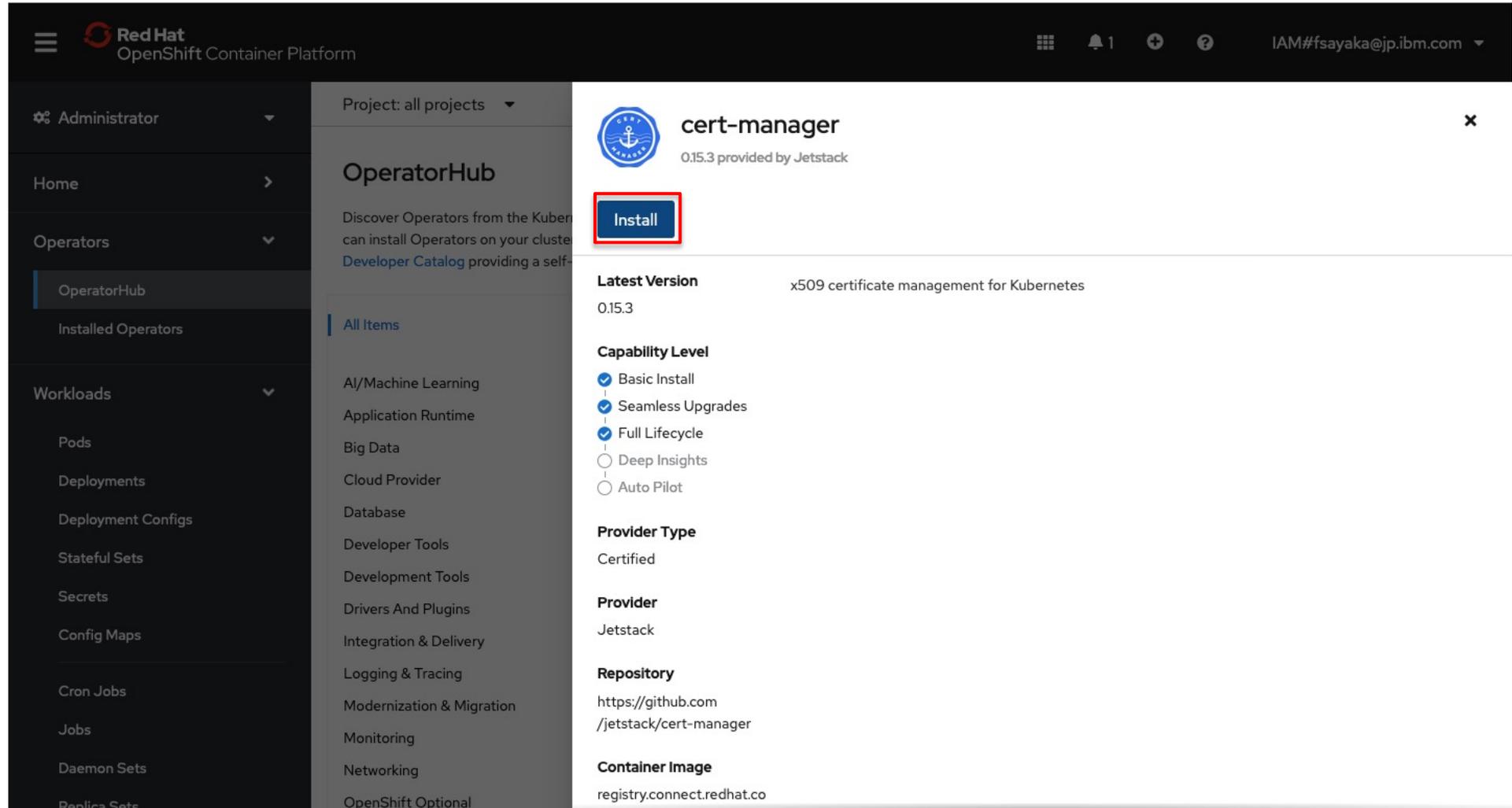
All Items

cert-manager 2 items

Operator	Provider	Marketplace
cert-manager	provided by Jetstack	
cert-manager	provided by Jetstack	Marketplace

## 2.1. OperatorによるGitLabのインストール – 4.Cert Manager の導入

### 4. 「cert-manager」の画面で「Install」をクリックする



The screenshot displays the Red Hat OpenShift Container Platform interface. The top navigation bar includes the Red Hat logo, the text "OpenShift Container Platform", and the user email "IAM#fsayaka@jp.ibm.com". The left sidebar shows the "OperatorHub" menu item selected. The main content area displays the details for the "cert-manager" operator, version 0.15.3, provided by Jetstack. The "Install" button is highlighted with a red box. Below the button, the operator's details are listed:

- Latest Version:** 0.15.3 (x509 certificate management for Kubernetes)
- Capability Level:** Basic Install, Seamless Upgrades, Full Lifecycle (all selected), Deep Insights, Auto Pilot (all unselected).
- Provider Type:** Certified
- Provider:** Jetstack
- Repository:** https://github.com/jetstack/cert-manager
- Container Image:** registry.connect.redhat.co

## 2.1. OperatorによるGitLabのインストール – 4.Cert Manager の導入

- 5.表示された画面で「Install」をクリックする
  - 更新の方法（自動または手動）は必要であれば変更する

The screenshot shows the 'Install Operator' page in the Red Hat OpenShift Container Platform. The page is titled 'Install Operator' and provides instructions on how to install an operator by subscribing to an update channel. The 'Update Channel' is set to 'stable', and the 'Installation Mode' is set to 'All namespaces on the cluster (default)'. The 'Installed Namespace' is set to 'openshift-operators'. The 'Approval Strategy' is set to 'Automatic'. The 'cert-manager' operator, provided by Jetstack, is selected. The 'Provided APIs' section shows 'CM CertManager' representing the cert-manager installation. The 'Install' button is highlighted with a red box.

OperatorHub > Operator Installation

### Install Operator

Install your Operator by subscribing to one of the update channels to keep the Operator up to date. The strategy determines either manual or automatic updates.

**Update Channel \***

- alpha
- stable

**Installation Mode \***

- All namespaces on the cluster (default)  
Operator will be available in all namespaces.
- A specific namespace on the cluster  
This mode is not supported by this Operator

**Installed Namespace \***

PR openshift-operators

**Approval Strategy \***

- Automatic
- Manual

**cert-manager**  
provided by Jetstack

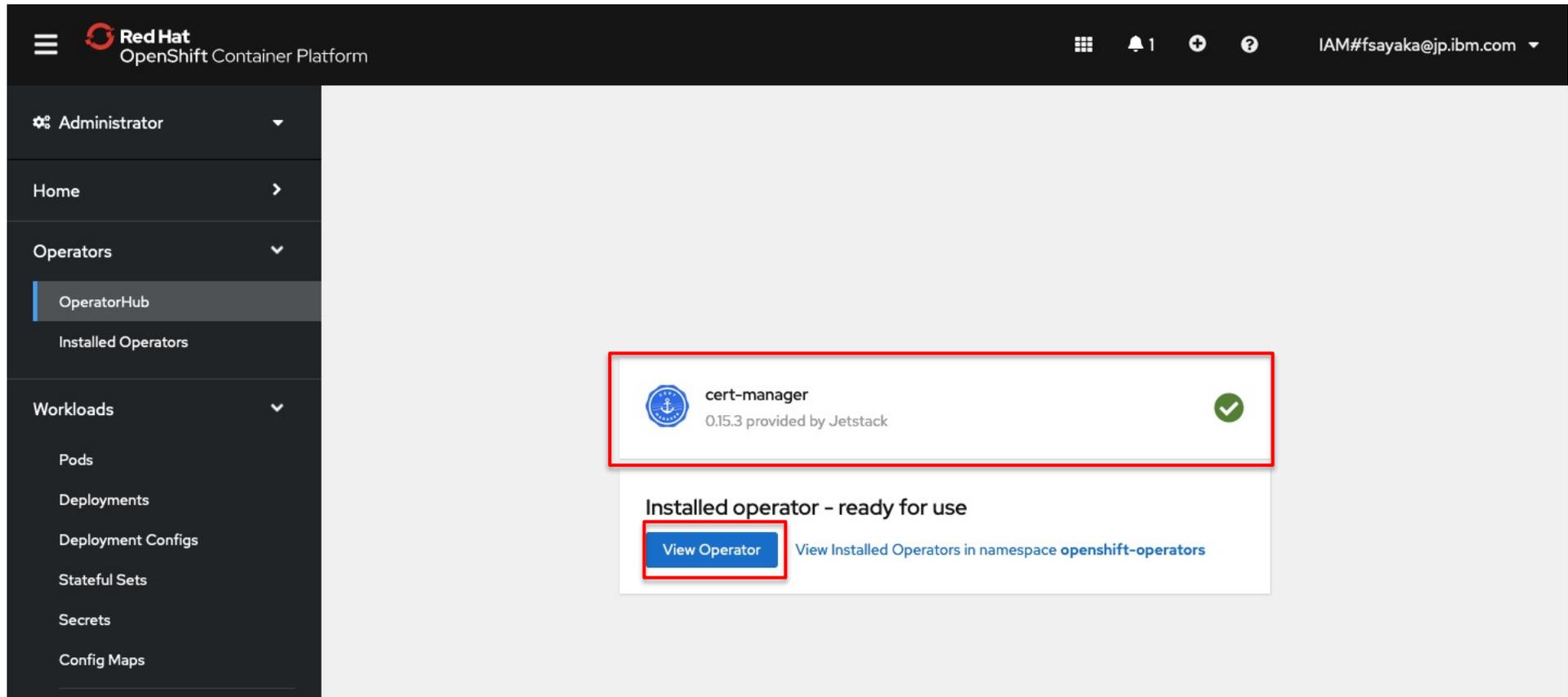
**Provided APIs**

CM CertManager  
Represents the cert-manager installation

**Install** **Cancel**

## 2.1. OperatorによるGitLabのインストール – 4.Cert Manager の導入

- 6.導入が完了すると以下のように緑色のチェックマークが表示される  
「View Operator」をクリックする



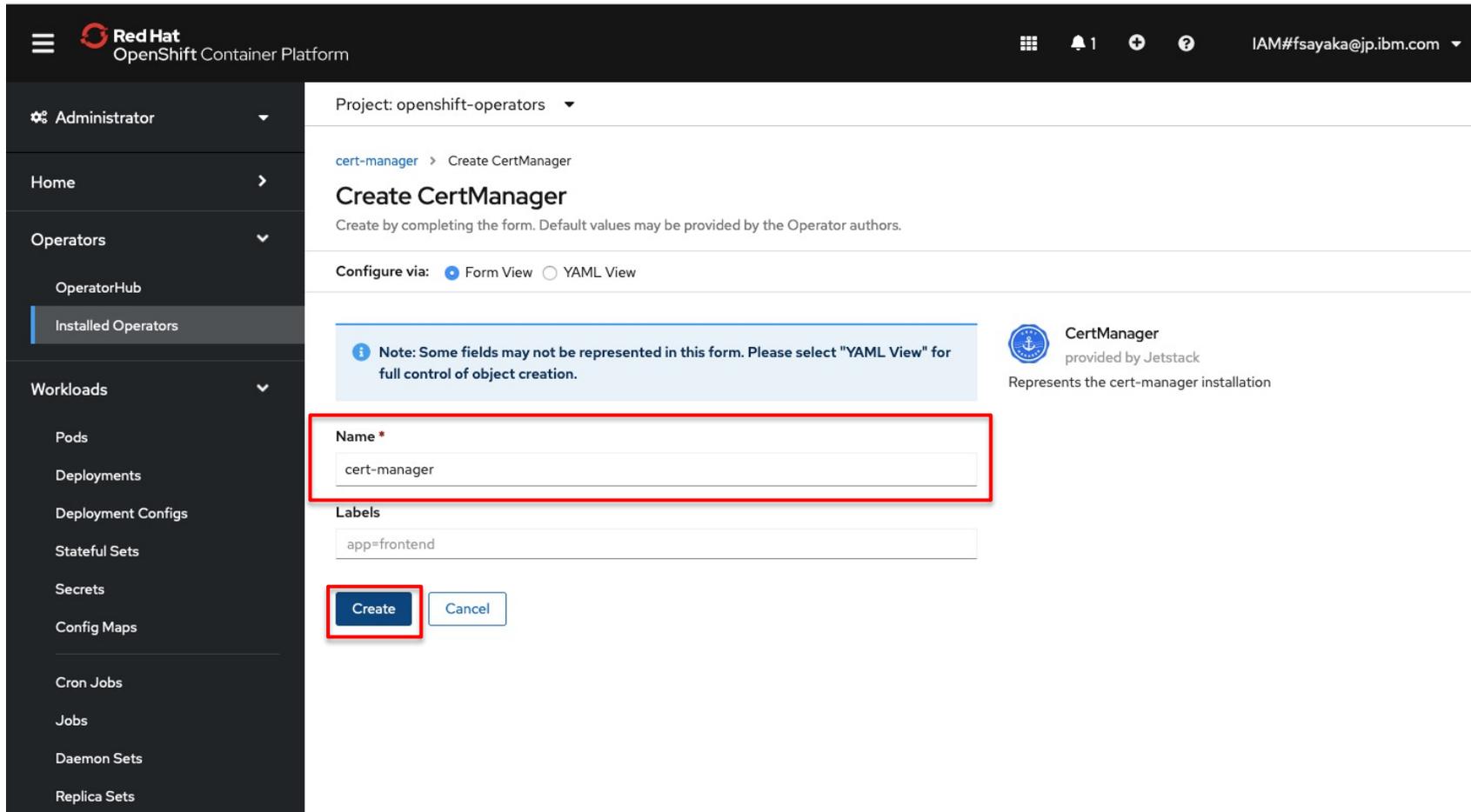
## 2.1. OperatorによるGitLabのインストール – 4.Cert Manager の導入

7. cert-manager Operatorの画面で「Create instance」をクリックする

The screenshot displays the Red Hat OpenShift Container Platform interface. The top navigation bar includes the Red Hat logo, the text 'OpenShift Container Platform', and user information 'IAM#fsayaka@jp.ibm.com'. The left sidebar shows a navigation menu with 'Administrators', 'Home', 'Operators', and 'Workloads'. Under 'Operators', 'Installed Operators' is selected. The main content area shows the details for the 'cert-manager' operator (version 0.15.3 provided by Jetstack). The 'Provided APIs' section contains a card for 'CertManager' with a 'Create Instance' button highlighted by a red box. The 'Description' section states 'x509 certificate management for Kubernetes'. The right sidebar lists metadata: 'Provider: Jetstack', 'Created At: less than a minute ago', 'Links: repository (https://github.com/jetstack/cert-manager), containerImage (https://quay.io/jetstack/cert-manager:latest)', and 'Maintainers: cert-manager team (cert-manager-maintainers@jetstack.io)'.

## 2.1. OperatorによるGitLabのインストール – 4.Cert Manager の導入

8. Nameに「cert-manager」と入力し、「Create」をクリックする



The screenshot shows the Red Hat OpenShift Container Platform interface. The left sidebar contains navigation options: Administrator, Home, Operators, Installed Operators, Workloads, Pods, Deployments, Deployment Configs, Stateful Sets, Secrets, Config Maps, Cron Jobs, Jobs, Daemon Sets, and Replica Sets. The main content area displays the 'Create CertManager' form. The 'Name' field is highlighted with a red box and contains the text 'cert-manager'. Below the 'Name' field is the 'Labels' field, which contains 'app=frontend'. At the bottom of the form, the 'Create' button is highlighted with a red box. A note above the form states: 'Note: Some fields may not be represented in this form. Please select "YAML View" for full control of object creation.' The 'CertManager' logo and name are visible on the right side of the form.

## 2.1. OperatorによるGitLabのインストール – 4.Cert Manager の導入

9. 「CertManager」タブに作成したcert-managerのインスタンスが表示される

The screenshot shows the OpenShift OperatorHub interface. The left sidebar contains navigation options: Administrator, Home, Operators (OperatorHub, Installed Operators), and Workloads (Pods, Deployments, Deployment Configs, Stateful Sets, Secrets, Config Maps, Cron Jobs, Jobs, Daemon Sets, Replica Sets). The main content area is for the 'cert-manager' operator (version 0.15.3 provided by Jetstack). The 'CertManager' tab is selected and highlighted with a red box. Below the tabs, there is a 'Create CertManager' button and a search bar. A table lists the installed CertManagers:

Name ↑	Kind ↓	Status ↑	Labels ↓
CM cert-manager	CertManager	Conditions: Initialized, Deployed	No labels

## 2.1. OperatorによるGitLabのインストール – 導入の流れ

導入の流れは以下の通り

1. 端末の準備
2. ドメイン名の準備 (default routeの確認)
3. NGINX Ingress の導入
4. Cert Manager の導入
5. OperatorとGitLabの導入
6. 接続確認

## 2.1. OperatorによるGitLabのインストール -5.OperatorとGitLabの導入

OperatorによるGitLabの導入は以下の流れで実施する

- Git operatorリポジトリのclone
- GitLabのCRD(Custom Resource Definition)のデプロイ
- GitLab Operatorのデプロイ
- GitLab Custom Resource 定義ファイルの作成
- GitLabインスタンスのデプロイ

### ■ Git operatorリポジトリのclone

1. Macの場合ターミナル、Windowsの場合はPowerShellを開く
2. Git リポジトリを作成するディレクトリへ移動 (cd) する
3. 以下のコマンドを実行してGit operatorリポジトリをcloneする

```
cd <Gitリポジトリ作成ディレクトリ>
```

```
git clone https://gitlab.com/gitlab-org/gl-openshift/gitlab-operator.git
```

4. cloneしたリポジトリのディレクトリに移動する

```
cd gitlab-operator
```

## 2.1. OperatorによるGitLabのインストール -5.OperatorとGitLabの導入

### ■ GitLabのCRD(Custom Resource Definition)のデプロイ

#### 1. GitLabを導入するOpenShiftクラスターにログインする

```
oc login --token=<トークン> --server=<接続先サーバーのURL>
```

– 実行例

```
$ oc login --token=xxx --server=https://c100-e.jp-tok.containers.cloud.ibm.com:32329  
Logged into "https://c100-e.jp-tok.containers.cloud.ibm.com:32329" as "IAM#xxx" using the token provided.  
  
You have access to 64 projects, the list has been suppressed. You can list all projects with 'oc projects'  
  
Using project "default".
```

## 2.1. OperatorによるGitLabのインストール -5.OperatorとGitLabの導入

### ■ GitLabのCRD(Custom Resource Definition)のデプロイ

#### 2. gitlab-operator ディレクトリーで以下のコマンドを実行する

```
make install_crds
```

– 実行例

```
$ make install_crds
go: creating new go.mod: module tmp
go get: added sigs.k8s.io/controller-tools v0.3.0
/Users/aha01107/go/bin/controller-gen "crd:trivialVersions=true,preserveUnknownFields=false"
rbac:roleName=manager-role webhook paths="..." output:crd:artifacts:config=config/crd/bases
/usr/local/bin/kustomize build config/crd | kubectl apply -f -
customresourcedefinition.apiextensions.k8s.io/gitlabs.apps.gitlab.com created
```

#### 3. GitLabのCRDが作成されたことを確認する

```
oc get crd | grep gitlab
```

– 実行例

```
$ oc get crd | grep gitlab
gitlabs.apps.gitlab.com                2021-04-23T01:28:06Z
```

## 2.1. OperatorによるGitLabのインストール -5.OperatorとGitLabの導入

### ■ GitLab Operatorのデプロイ

#### 1. gitlab-operator ディレクトリーで以下のコマンドを実行する

```
make deploy_operator
```

#### – 実行例

```
$ make deploy_operator
go: creating new go.mod: module tmp
go get: added sigs.k8s.io/controller-tools v0.3.0
/Users/aha01107/go/bin/controller-gen "crd:trivialVersions=true,preserveUnknownFields=false" rbac:roleName=manager-role webhook paths="/..."
output:crd:artifacts:config=config/crd/bases
cd config/manager && /usr/local/bin/kustomize edit set image registry.gitlab.com/gitlab-org/gl-openshift/gitlab-operator=registry.gitlab.com/gitlab-org/gl-openshift/gitlab-operator:latest
cd config/manager && /usr/local/bin/kustomize edit add patch --path patches/deployment_always_pull_image.yaml
2021/04/23 16:18:22 patch types.Patch{Path:"patches/deployment_always_pull_image.yaml", Patch:"", Target:(*types.Selector)(nil)} already in kustomization file
cd config/default && /usr/local/bin/kustomize edit set namespace gitlab-system
/usr/local/bin/kustomize build config/default | kubectl apply -f -
namespace/gitlab-system created
customresourcedefinition.apiextensions.k8s.io/gitlabs.apps.gitlab.com configured
serviceaccount/gitlab-app created
serviceaccount/gitlab-manager created
role.rbac.authorization.k8s.io/gitlab-leader-election-role created
clusterrole.rbac.authorization.k8s.io/gitlab-app-role created
clusterrole.rbac.authorization.k8s.io/gitlab-manager-role created
clusterrole.rbac.authorization.k8s.io/gitlab-proxy-role created
clusterrole.rbac.authorization.k8s.io/gitlab-metrics-reader created
rolebinding.rbac.authorization.k8s.io/gitlab-leader-election-rolebinding created
clusterrolebinding.rbac.authorization.k8s.io/gitlab-gitlab-app-rolebinding created
clusterrolebinding.rbac.authorization.k8s.io/gitlab-gitlab-manager-rolebinding created
clusterrolebinding.rbac.authorization.k8s.io/gitlab-gitlab-proxy-rolebinding created
service/gitlab-controller-manager-metrics-service created
service/gitlab-webhook-service created
deployment.apps/gitlab-controller-manager created
certificate.cert-manager.io/gitlab-serving-cert created
issuer.cert-manager.io/gitlab-selfsigned-issuer created
validatingwebhookconfiguration.admissionregistration.k8s.io/gitlab-gitlab-validating-webhook-configuration created
```

## 2.1. OperatorによるGitLabのインストール -5.OperatorとGitLabの導入

### ■ GitLab Operatorのデプロイ

2. GitLab Operatorがデプロイされたプロジェクト“gitlab-system”に切り替える

```
oc project gitlab-system
```

3. 以下のコマンドを実行し、NAMEが「gitlab-controller-manager」で始まるGitLab OperatorのpodのREADYが2/2(コンテナがすべてがreadyである)、STATUSが「Running」になっていることを確認する

```
oc get pod
```

– 実行例

```
$ oc get pod
NAME                                READY STATUS  RESTARTS  AGE
gitlab-controller-manager-cf6f445c6-r2c9x 2/2  Running  1         115s
```

## 2.1. OperatorによるGitLabのインストール -5.OperatorとGitLabの導入

### ■ GitLab Custom Resource 定義ファイルの作成

1. エディターでyamlファイル（例、mygitlab.yaml）を作成し、以下の内容を記載する（ガイド作成時の例は次ページ参照）

```

apiVersion: apps.gitlab.com/v1beta1
kind: GitLab
metadata:
  name: example
spec:
  chart:
    version: "X.Y.Z" # select a version from the CHART_VERSIONS
    file in the root of this project
  values:
    global:
      hosts:
        domain: example.com # use a real domain here
    ingress:
      class: nginx # ensure this matches the ingress class defined
      within the NGINX ingress controller
      configureCertmanager: true
      certmanager-issuer:
        email: youremail@example.com # use your real email address
      here

```

- metadata.name
  - 設定したい名前を指定する
- spec.chart.version
  - 使用するGitLabのHelm chartのバージョンを指定する。選択可能な値は、gitlab-operatorディレクトリーのCHART\_VERSIONSファイルの内容を確認する
- spec.chart.values.global.hosts.domain
  - 2.1. OperatorによるGitLabのインストール – 2.ドメイン名の準備で用意したドメイン名を指定する
- その他の設定値については、[GitLab Helm Chart documentation](#) を参照のこと。

## 2.1. OperatorによるGitLabのインストール -5.OperatorとGitLabの導入

### ■ GitLab Custom Resource 定義ファイルの作成

#### ガイド作成時のyaml例

- metadata.name
  - 設定したい名前(gitlab)を記載
- spec.chart.values.global.hosts.domain
  - 2.1. OperatorによるGitLabのインストール - 2.ドメイン名の準備で確認したデフォルトのルーターのCanonical Host nameを記載
- spec.chart.version
  - 使用するGitLabのHelm chartのバージョンを記載

```
apiVersion: apps.gitlab.com/v1beta1
kind: GitLab
metadata:
  name: gitlab
  namespace: gitlab-system
spec:
  chart:
    values:
      global:
        hosts:
          domain: mycluster-tok02-b-889642-d85540d5ac7eb4665421ec4a86b8d368-0000.jp-
tok.containers.appdomain.cloud
        hostSuffix:
          version: 4.10.3
```

## 2.1. OperatorによるGitLabのインストール -5.OperatorとGitLabの導入

### ■ GitLabインスタンスのデプロイ

1.以下のコマンドを実行し、GitLabのインスタンスをデプロイする

```
oc -n gitlab-system apply -f <GitLab Custom Resource 定義ファイルの作成で作成したYAMLファイル名>
```

– 実行例

```
$ oc -n gitlab-system apply -f mygitlab.yaml
```

```
gitlab.apps.gitlab.com/gitlab created
```

2.以下のコマンドで、インストール実行中のOperatorのログを確認する

```
oc -n gitlab-system logs deployment/gitlab-controller-manager -c manager -f
```

## 2.1. OperatorによるGitLabのインストール -5.OperatorとGitLabの導入

### ■ GitLabインスタンスのデプロイ

3.インストールが終わると、以下のコマンドでGitLabリソースのSTATUSがRunningになる

```
oc get gitlabs -n gitlab-system
```

– 実行例

```
$ oc get gitlabs -n gitlab-system  
NAME STATUS  
gitlab Running
```

## 2.1. OperatorによるGitLabのインストール -5.OperatorとGitLabの導入

### ■ GitLabインスタンスのデプロイ

4.GitLabの各コンポーネントのpodの状況は以下のコマンドで確認する

```
oc get pod -n gitlab-system
```

- 実行例

```
$ oc get pod -n gitlab-system
NAME                                READY STATUS   RESTARTS AGE
gitlab-bucket-mljjh                 0/1   Completed 0       4m58s
gitlab-controller-manager-59bb6d7495-pwtqh 2/2   Running  0       36m
gitlab-gitaly-0                      1/1   Running  0       9m32s
gitlab-gitlab-exporter-79b5757597-b6pcz   1/1   Running  0       4m58s
gitlab-gitlab-shell-6b88fd658d-mbkxz     1/1   Running  0       4m58s
gitlab-migrations-1-x2sll            0/1   Completed 0       4m58s
gitlab-minio-0                       1/1   Running  0       9m32s
gitlab-postgresql-0                  2/2   Running  0       9m32s
gitlab-redis-master-0                2/2   Running  0       9m32s
gitlab-registry-5b495c7899-f6cgs        1/1   Running  0       4m58s
gitlab-registry-5b495c7899-rf62x        1/1   Running  0       4m58s
gitlab-shared-secrets-1-pb2-5f2q9       0/1   Completed 0       14m
gitlab-shared-secrets-1-ycx-selfsign-wp6ks 0/1   Completed 0       10m
gitlab-sidekiq-all-in-1-v1-6685f45688-nr6v6 1/1   Running  0       4m58s
gitlab-task-runner-95f78cbb6-pf95p      1/1   Running  0       4m58s
gitlab-webservice-default-84dc779c77-k256z 2/2   Running  0       4m58s
gitlab-webservice-default-84dc779c77-tvtnf 2/2   Running  0       4m58s
```

## 2.1. OperatorによるGitLabのインストール -5.OperatorとGitLabの導入

### ■ GitLabインスタンスのデプロイ

#### 5.GitLabのrouteを確認する

NAMEが「gitlab-webservice-default」で始まり、PATHが「/」のrouteのHOST/PORTの値が、外部からアクセスできるURLであるため控えておく

```
oc get route -n gitlab-system
```

#### – 実行例

- HOST/PORTの値「**gitlab.mycluster-tok02-b-889642-d85540d5ac7eb4665421ec4a86b8d368-0000.jp-tok.containers.appdomain.cloud**」を控えておく。

```
$ oc get route -n gitlab-system
```

NAME	HOST/PORT	TERMINATION	WILDCARD	PATH	
gitlab-minio-wwcgh	minio.mycluster-tok02-b-889642-d85540d5ac7eb4665421ec4a86b8d368-0000.jp-tok.containers.appdomain.cloud	/	gitlab-minio	minio edge/Redirect	None
gitlab-registry-tf2bp	registry.mycluster-tok02-b-889642-d85540d5ac7eb4665421ec4a86b8d368-0000.jp-tok.containers.appdomain.cloud	/	gitlab-registry	registry edge/Redirect	None
gitlab-webservice-default-2nfnj	<b>gitlab.mycluster-tok02-b-889642-d85540d5ac7eb4665421ec4a86b8d368-0000.jp-tok.containers.appdomain.cloud</b>	/	gitlab-webservice-default	http-	
workhorse	edge/Redirect	None			
gitlab-webservice-default-j92d9	<b>gitlab.mycluster-tok02-b-889642-d85540d5ac7eb4665421ec4a86b8d368-0000.jp-tok.containers.appdomain.cloud</b>	/admin/sidekiq/	gitlab-webservice-default	http-	
webservice	edge/Redirect	None			

## 2.1. OperatorによるGitLabのインストール – 導入の流れ

導入の流れは以下の通り

1. 端末の準備
2. ドメイン名の準備 (default routeの確認)
3. NGINX Ingress の導入
4. Cert Manager の導入
5. OperatorとGitLabの導入
6. **接続確認**

## 2.1. OperatorによるGitLabのインストール – 6. 接続確認

### 1. 以下のコマンドでrootユーザーのパスワードを確認する

- <name>には「2.1. OperatorによるGitLabのインストール -5.OperatorとGitLabの導入」GitLab Custom Resource 定義ファイルの作成 で指定した metadata.name の値を指定する（本ガイドの例では gitlab）

```
oc get secret <name>-gitlab-initial-root-password -ojsonpath='{.data.password}' | base64 --decode ; echo
```

- 参考 : GitLabマニュアル- Deployment Guide > Initial login

<https://docs.gitlab.com/charts/installation/deployment.html#initial-login>

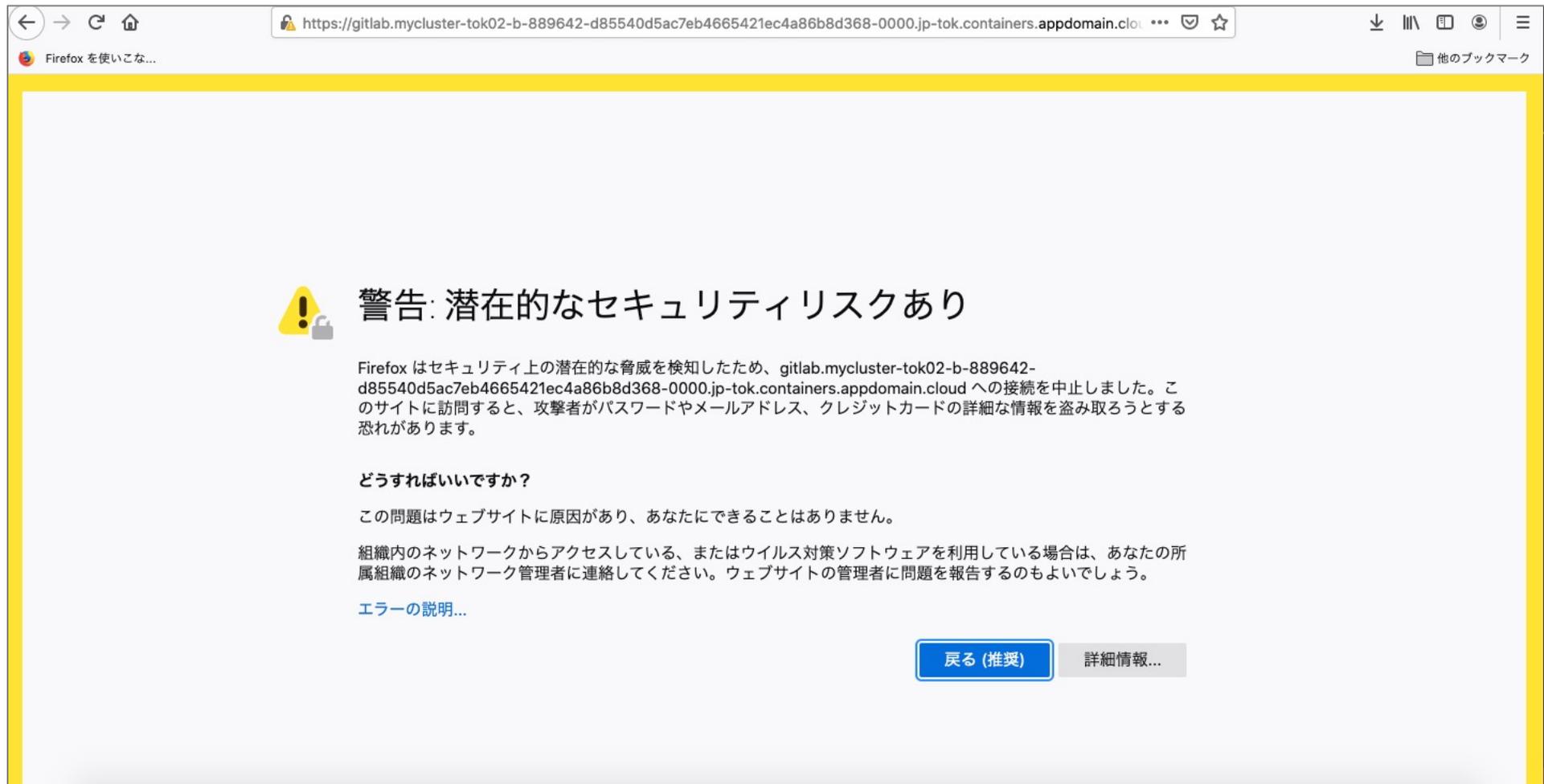
## 2.1. OperatorによるGitLabのインストール – 6. 接続確認

2. ブラウザーを開き、「2.1. OperatorによるGitLabのインストール -5.OperatorとGitLabの導入、GitLabインスタンスのデプロイ、5.GitLabのrouteを確認する」で確認したrouteのURLへアクセスする



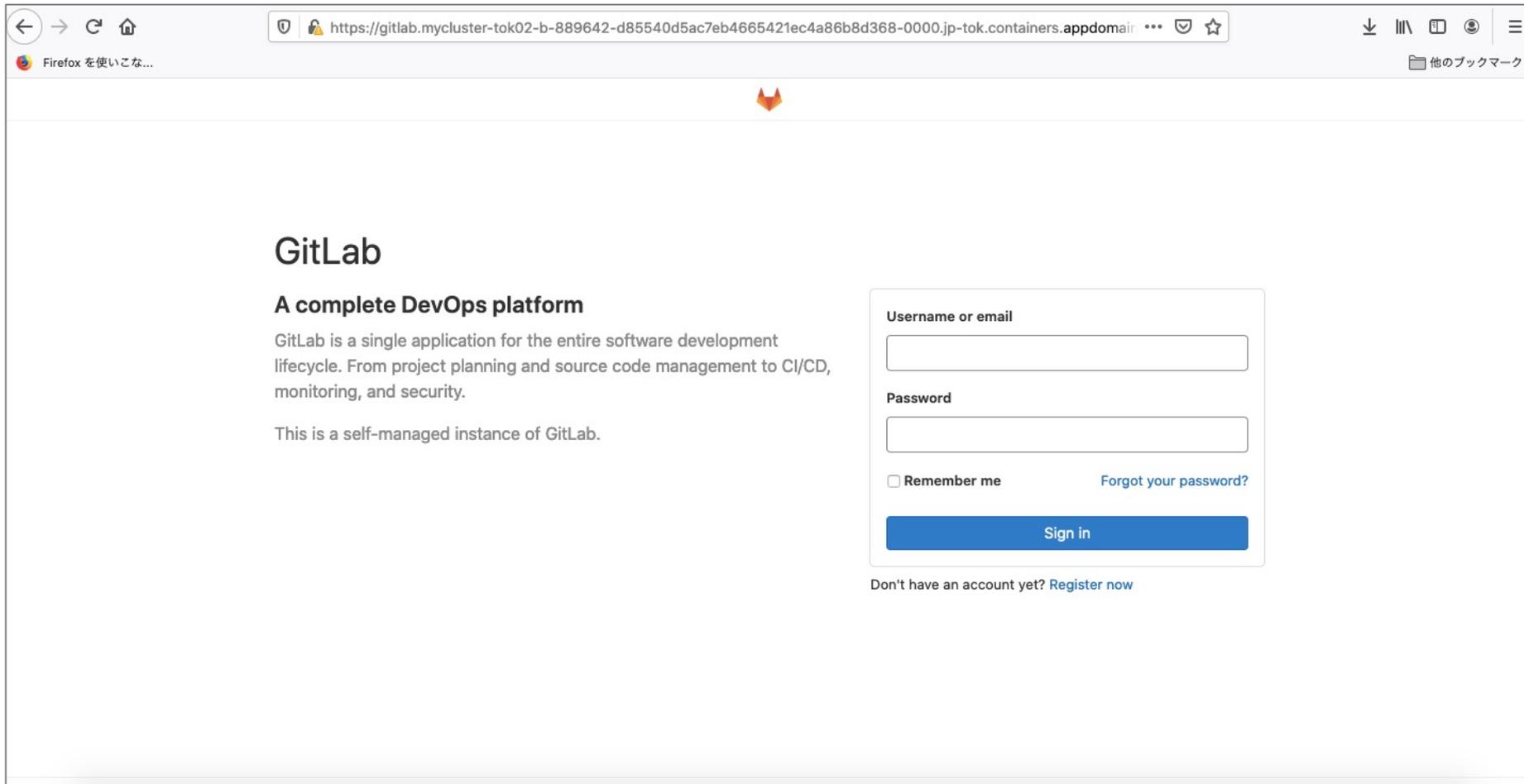
## 2.1. OperatorによるGitLabのインストール – 6. 接続確認

3. 自己署名証明書を利用している場合、ブラウザーによる警告が表示される  
信頼する証明書として登録するか、危険性を理解して先へ進む



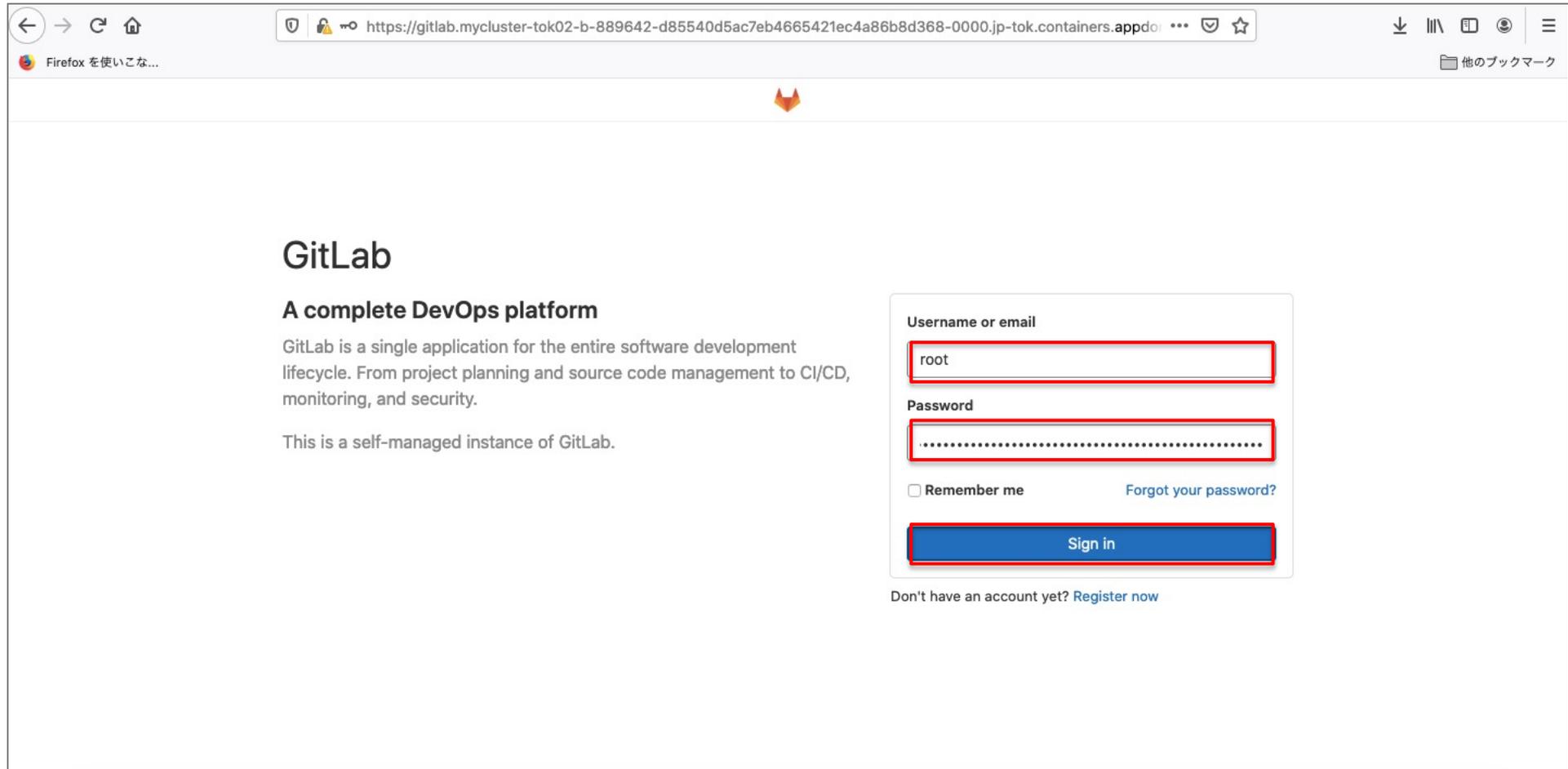
## 2.1. OperatorによるGitLabのインストール – 6. 接続確認

### 4. GitLabのログイン画面が表示されることを確認する



## 2.1. OperatorによるGitLabのインストール – 6. 接続確認

5. Username or email欄に「root」、Password欄に1. で確認したrootのパスワードを入力し、「Sign in」をクリックする



Firefox を使いこな...

https://gitlab.mycluster-tok02-b-889642-d85540d5ac7eb4665421ec4a86b8d368-0000.jp-tok.containers.appdo...

# GitLab

## A complete DevOps platform

GitLab is a single application for the entire software development lifecycle. From project planning and source code management to CI/CD, monitoring, and security.

This is a self-managed instance of GitLab.

Username or email  
root

Password  
.....

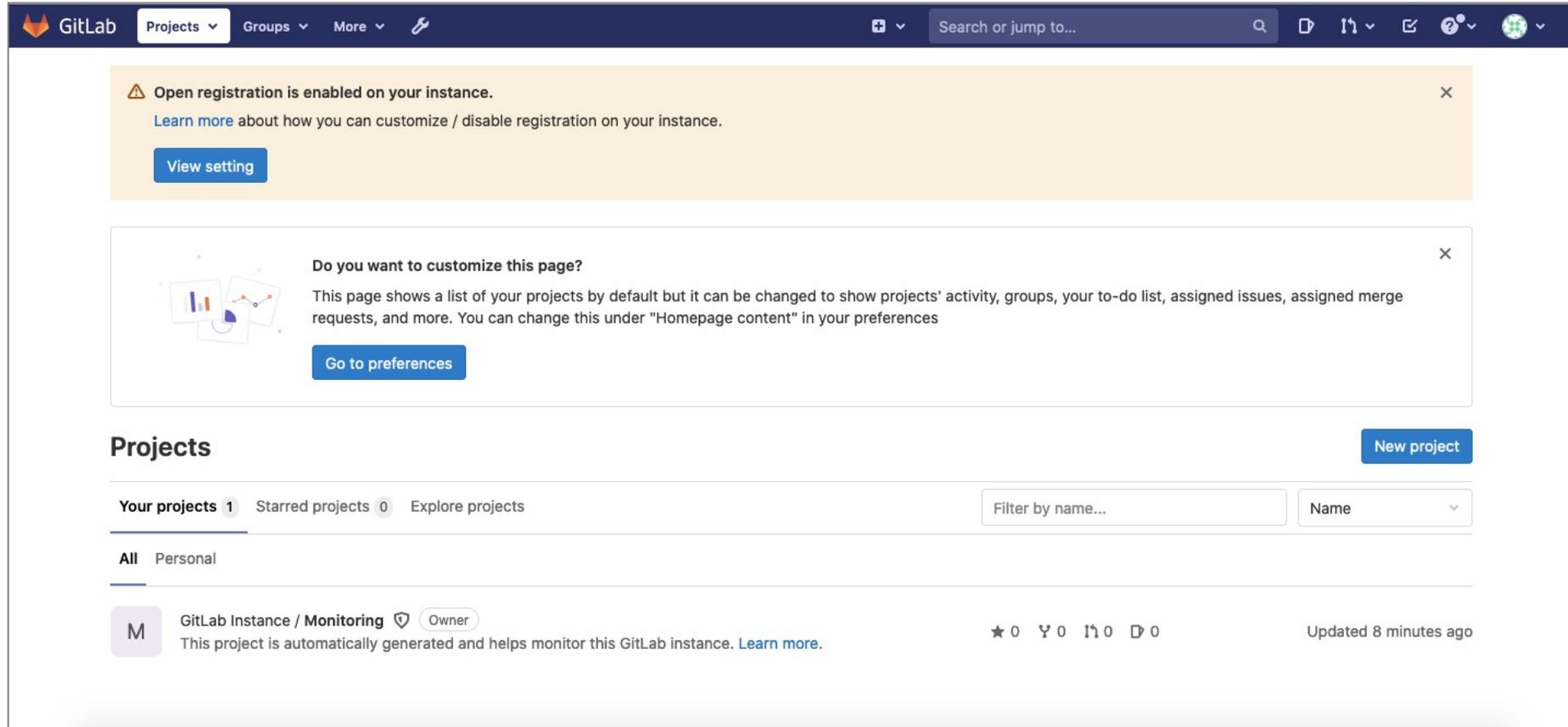
Remember me [Forgot your password?](#)

[Sign in](#)

Don't have an account yet? [Register now](#)

## 2.1. OperatorによるGitLabのインストール – 6. 接続確認

### 6. GitLabの画面が表示されることを確認する



以降、GitLab画面については、以下の手順でGUIを日本語にした状態での手順を記載するため、利用するユーザーで以下の手順を実施する

- GitLab GUI の右上の自分のアイコン → Preferences → Localization → Language を Japanese 日本語にする

2.2. Operator による GitLab Runner のインストールと設定

## 2. インストール

# インストールの流れ

- 以下の手順でインストールを行う

**1**

OperatorによるGitLabのインストール

**2**

OperatorによるGitLab Runnerのインストール

**3**

ライセンスの設定

## 2.2 OperatorによるGitLab Runnerのインストール

- 通常、OpenShiftでOperatorによるインストールを行う場合、OpenShiftのWebコンソールを利用する
- 本ガイド作成時点(2021年6月末)は、GitLab RunnerのOperatorはOpenShiftのWebコンソールから利用可能であるため、以下のリンク先を参考にOpenShiftのWebコンソールを使用したインストール手順を説明する

参考 : GitLab マニュアル - Install GitLab Runner on Red Hat OpenShift

<https://docs.gitlab.com/runner/install/openshift.html>

## 2.2 OperatorによるGitLab Runnerのインストール – 導入の流れ

導入の流れは以下の通り

1. GitLab Runner Operatorの導入
2. GitLabRunner登録トークン、URLの確認とsecretの作成
3. (自己署名証明書を使用する場合) route証明書の確認とsecretの作成
4. GitLab Runnerの導入
5. 接続確認
6. 補足設定

## 2.2 OperatorによるGitLab Runnerのインストール – 導入の流れ

導入の流れは以下の通り

1. **GitLab Runner Operatorの導入**
2. GitLabRunner登録トークン、URLの確認とsecretの作成
3. (自己署名証明書を使用する場合) route証明書の確認とsecretの作成
4. GitLab Runnerの導入
5. 接続確認
6. 補足設定

## 2.2 OperatorによるGitLab Runnerのインストール – 1. GitLab Runner Operatorの導入

### ■ GitLab RunnerのOperatorを導入する

1. 管理者権限のあるユーザーでOpenShiftコンソールにログインし、左側のメニューから「Operators」を展開し「OperatorHub」をクリックする

The screenshot shows the OpenShift OperatorHub console. The top navigation bar includes the Red Hat OpenShift Container Platform logo, a hamburger menu, and the user's email address (IAM#fsayaka@jp.ibm.com). The left sidebar contains a navigation menu with 'Operators' and 'OperatorHub' highlighted with red boxes. The main content area displays the 'OperatorHub' page, which includes a search bar with 'Nginx' entered and a list of operators. Two operators are visible: 'APIcast' (provided by Red Hat) and 'Nginx Ingress Operator' (provided by NGINX Inc).

Red Hat OpenShift Container Platform

Administrator

Home

Operators

OperatorHub

Installed Operators

Workloads

Pods

Deployments

Deployment Configs

Stateful Sets

Secrets

Config Maps

Cron Jobs

Project: all projects

### OperatorHub

Discover Operators from the Kubernetes community and Red Hat partners, curated by Red Hat. You can purchase commercial software through [Red Hat Marketplace](#). You can install Operators on your clusters to provide optional add-ons and shared services to your developers. After installation, the Operator capabilities will appear in the [Developer Catalog](#) providing a self-service experience.

All Items

All Items

2 items

API/Machine Learning

Application Runtime

Big Data

Cloud Provider

Database

Developer Tools

Development Tools

Drivers And Plugins

Integration & Delivery

Logging & Tracing

Community

APIcast  
provided by Red Hat

APIcast is an API gateway built on top of NGINX. It is part of the 3scale API Management Platform

Nginx Ingress Operator  
provided by NGINX Inc

The NGINX Ingress Operator is a Kubernetes/OpenShift component which deploys and...

## 2.2 OperatorによるGitLab Runnerのインストール – 1. GitLab Runner Operatorの導入

### 2. All itemsの下の欄に「GitLab Runner」と入力する

The screenshot shows the Red Hat OpenShift Container Platform OperatorHub interface. The search bar under the 'All Items' section contains the text 'GitLab Runner', which is highlighted with a red box. Below the search bar, two search results for 'GitLab Runner' are displayed, each provided by GitLab, Inc. The interface includes a sidebar with navigation options like Administrator, Home, and Operators, and a top navigation bar with the Red Hat logo and user information.

## 2.2 OperatorによるGitLab Runnerのインストール – 1. GitLab Runner Operatorの導入

### 3. 表示された「GitLab Runner」（Communityの記載のない方）をクリックする

The screenshot shows the Red Hat OpenShift Container Platform interface. The top navigation bar includes the Red Hat logo, the text 'Red Hat OpenShift Container Platform', and user information 'IAM#fsayaka@jp.ibm.com'. The left sidebar contains navigation options: Administrator, Home (with sub-items: Overview, Projects, Search, Explore, Events), Operators (with sub-items: OperatorHub, Installed Operators), Workloads, and Networking. The main content area is titled 'OperatorHub' and includes a search bar with 'GitLab Runner' entered. Below the search bar, two operator cards are displayed. The first card, which is highlighted with a red border, is for 'GitLab Runner provided by GitLab, Inc.' and is not marked as 'Community'. The second card is also for 'GitLab Runner provided by GitLab, Inc.' but is marked as 'Community'. The text on both cards reads: 'GitLab Runner operator manages lifecycle of GitLab Runner instances'. The search results show '2 items'.

## 2.2 OperatorによるGitLab Runnerのインストール – 1. GitLab Runner Operatorの導入

### 4. Operatorの詳細ページで「install」をクリックする

The screenshot shows the Red Hat OpenShift Container Platform interface. The left sidebar contains navigation options: Administrator, Home, Overview, Projects, Search, Explore, Events, Operators (selected), Installed Operators, Workloads, and Networking. The main content area displays the OperatorHub for GitLab Runner. The 'Install' button is highlighted with a red box. Below the button, the page provides details about the operator, including the latest version (13.9.0-rc2), capability level (Basic Install, Seamless Upgrades, Full Lifecycle, Deep Insights, Auto Pilot), provider type (Certified), provider (GitLab, Inc.), and repository.

**Red Hat OpenShift Container Platform**

Administrator

Home

Overview

Projects

Search

Explore

Events

Operators

OperatorHub

Installed Operators

Workloads

Networking

Project: all projects

**OperatorHub**

Discover Operators from the Kubernetes Operator Catalog that can be installed on your cluster. Visit the [Developer Catalog](#) providing a self-service interface for installing Operators.

All Items

AI/Machine Learning

Application Runtime

Big Data

Cloud Provider

Database

Developer Tools

Development Tools

Drivers And Plugins

Integration & Delivery

Logging & Tracing

**GitLab Runner**

13.9.0-rc2 provided by GitLab, Inc.

**Install**

**Latest Version**

13.9.0-rc2

**Capability Level**

- Basic Install
- Seamless Upgrades
- Full Lifecycle
- Deep Insights
- Auto Pilot

**Provider Type**

Certified

**Provider**

GitLab, Inc.

**Repository**

gitlab/gitlab-runner

GitLab Runner is the lightweight, highly-scalable agent that runs your build jobs and sends the results back to a GitLab instance. GitLab Runner works in conjunction with GitLab CI/CD, the open-source continuous integration service included with GitLab.

The GitLab Runner operator manages the lifecycle of GitLab Runner in Kubernetes or Openshift clusters. The operator aims to automate the tasks needed to run your CI/CD jobs in your container orchestration platform.

**Usage**

To link a GitLab Runner instance to a self-hosted GitLab instance or the hosted [GitLab](#), you first need to:

- create a secret containing the `runner-registration-token` key.
 

```
$ kubectl create secret generic runner-token-secret --from-literal runner-registration-token="TOKEN_FROM_GITLAB_INSTANCE"
```
- Create a Runner instance. Notice how the secret name is referenced in the example below.

## 2.2 OperatorによるGitLab Runnerのインストール – 1. GitLab Runner Operatorの導入

- 5. Install Operator ページで以下のように設定し、「Install」をクリックする

OperatorHub > Operator Installation

### Install Operator

Install your Operator by subscribing to one of the update channels to keep the Operator up to date. The strategy determines either manual or automatic updates.

**Update Channel \***

- beta
- stable

**Installation Mode \***

- All namespaces on the cluster (default)  
This mode is not supported by this Operator
- A specific namespace on the cluster  
Operator will be available in a single namespace only.

**Installed Namespace \***

PR gitlab-system

**Approval Strategy \***

- Automatic
- Manual

**GitLab Runner**  
provided by GitLab, Inc.

**Provided APIs**

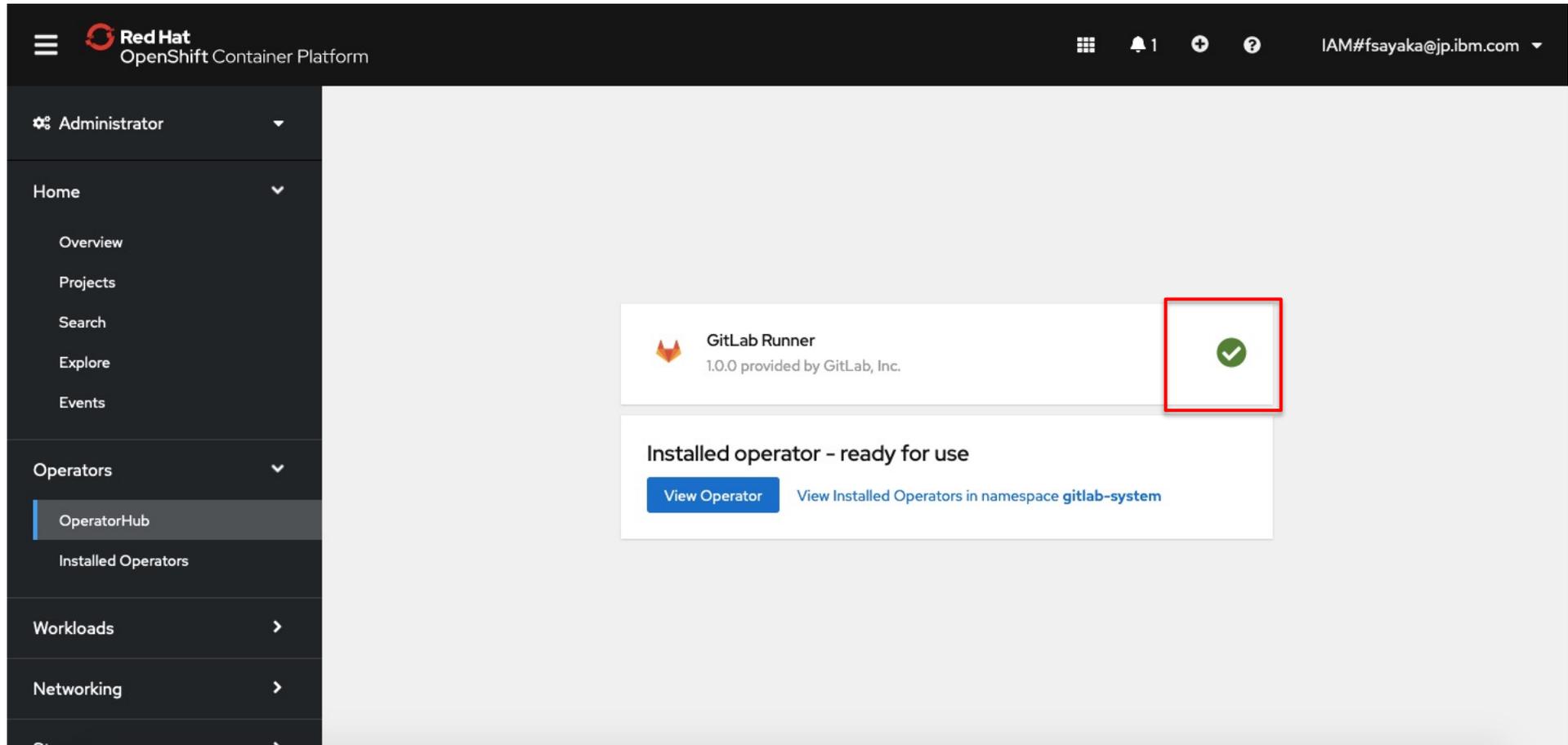
- R GitLab Runner**  
Runner is the open source project used to run your jobs and send the results back to GitLab

**Install** **Cancel**

- Update Channel で「stable」を選択
- Installed Namespace で導入先のプロジェクトを選択

## 2.2 OperatorによるGitLab Runnerのインストール – 1. GitLab Runner Operatorの導入

6. インストールが完了すると以下のように緑色のチェックマークが表示される



## 2.2 OperatorによるGitLab Runnerのインストール – 1. GitLab Runner Operatorの導入

7. 左側の「Operators」 > 「Installed Operators」を選択し、Projectのリストから導入先のプロジェクトを選択する  
GitLab Runner のStatusが「Succeeded」となっていればOperatorは準備完了となっている

The screenshot shows the Red Hat OpenShift Container Platform interface. The left sidebar has 'Installed Operators' selected under the 'Operators' section. The main content area shows the 'Installed Operators' page for the 'gitlab-system' project. A table lists the installed operators:

Name	Managed Namespaces	Status	Provided APIs
<b>GitLab Runner</b> 1.0.0 provided by GitLab, Inc.	NS gitlab-system	✓ Succeeded Up to date	GitLab Runner
<b>cert-manager</b> 1.1.0 provided by Jetstack	All Namespaces	✓ Succeeded Up to date	CertManager

## 2.2 OperatorによるGitLab Runnerのインストール – 導入の流れ

導入の流れは以下の通り

1. GitLab Runner Operatorの導入
2. **GitLabRunner登録トークン、URLの確認とsecretの作成**
3. (自己署名証明書を使用する場合) route証明書の確認とsecretの作成
4. GitLab Runnerの導入
5. 接続確認
6. 補足設定

## 2.2 OperatorによるGitLab Runnerのインストール -2. GitLabRunner登録トークン、URLの確認とsecretの作成

### ■ Runnerの登録に使用する登録トークンとURLを確認する

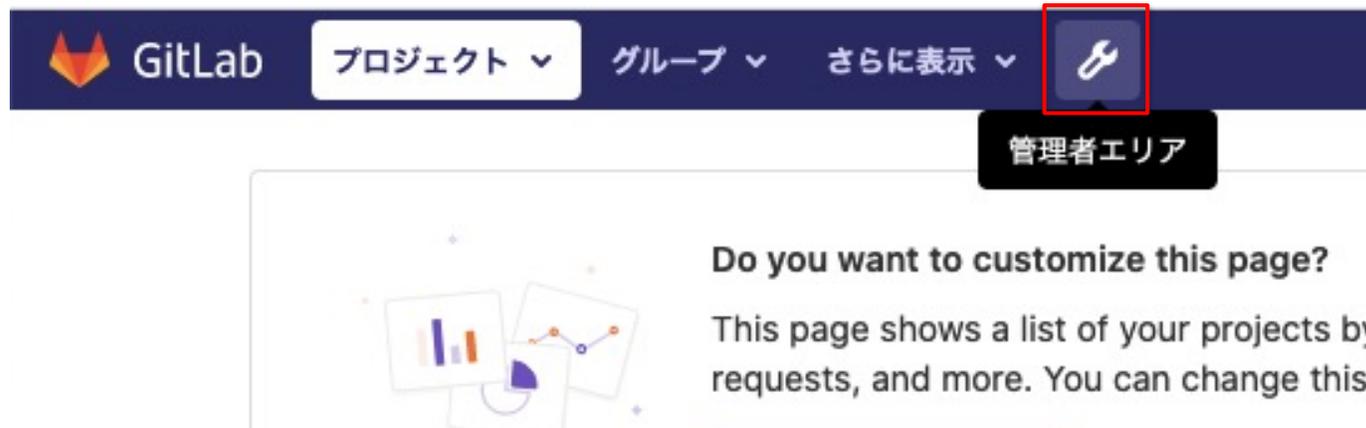
Runnerの種類によって確認方法が異なる

- 共有Runnerの場合（GitLabインスタンス全体で共有するRunner）
  - 管理エリアの「概要>Runner」画面で確認する
- グループRunner（グループ内のプロジェクトで共有するRunner）
  - グループの「設定 > CI/CD > Runner」で確認する
- プロジェクト固有のRunner（プロジェクト内で利用するRunner）
  - プロジェクトの「設定 > CI/CD > Runner」で確認する

本ガイドでは、以降共有Runnerを利用する場合の手順を説明する

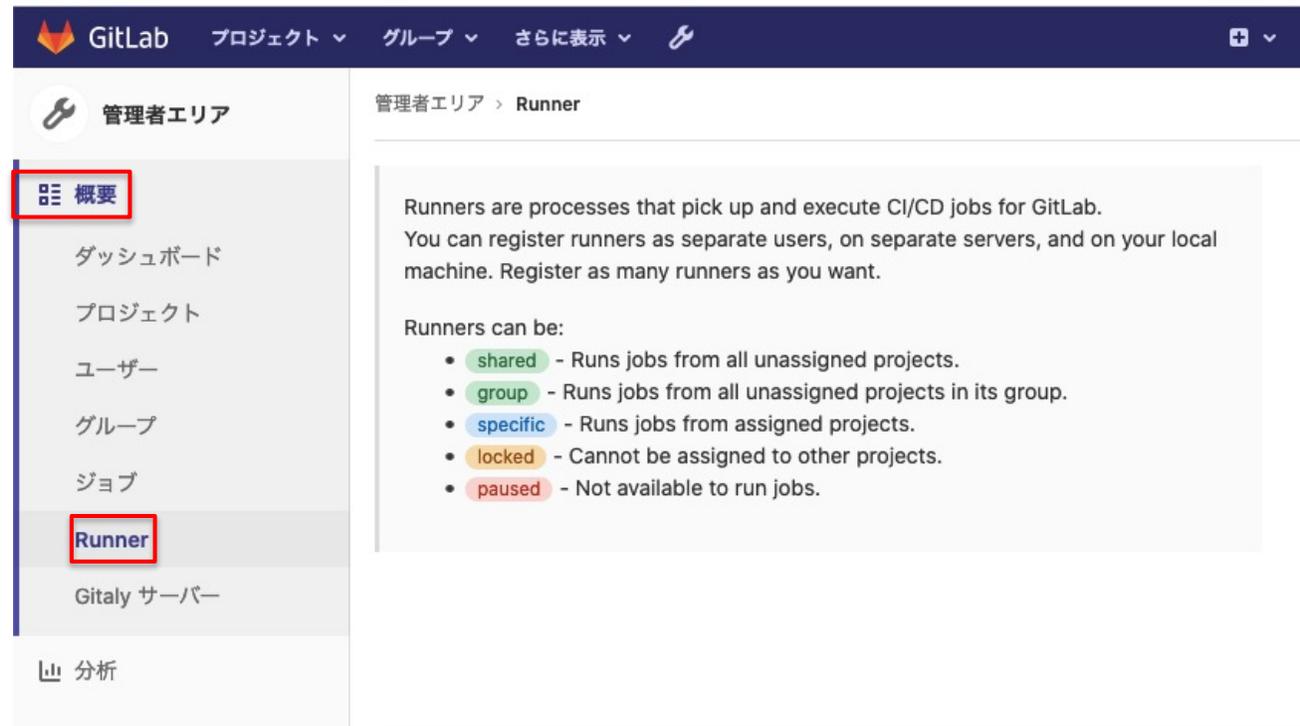
### ■ 共有Runner用の登録トークンとURLの確認

1. GitLabに管理者としてログインし、管理者エリアアイコンをクリックする



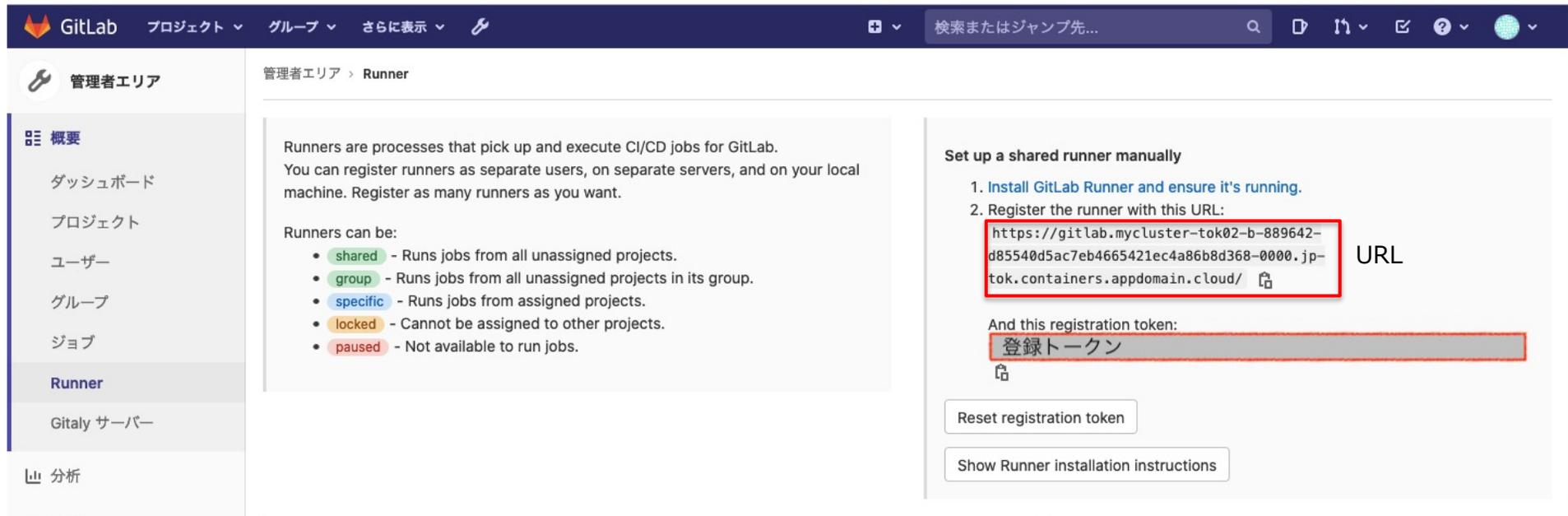
## 2.2 OperatorによるGitLab Runnerのインストール - 2. GitLabRunner登録トークン、URLの確認とsecretの作成

2.管理者エリアの左側の「概要」の下「Runner」をクリックする



## 2.2 OperatorによるGitLab Runnerのインストール -2. GitLabRunner登録トークン、URLの確認とsecretの作成

### 3.表示された登録トークンとURLを控えておく



The screenshot shows the GitLab Admin Area for the 'Runner' section. The left sidebar contains navigation options: 管理者エリア (Admin Area), 概要 (Overview), ダッシュボード (Dashboard), プロジェクト (Project), ユーザー (User), グループ (Group), ジョブ (Job), Runner (selected), Gitaly サーバー (Gitaly Server), and 分析 (Analytics). The main content area is titled 'Runner' and contains the following information:

Runners are processes that pick up and execute CI/CD jobs for GitLab. You can register runners as separate users, on separate servers, and on your local machine. Register as many runners as you want.

Runners can be:

- **shared** - Runs jobs from all unassigned projects.
- **group** - Runs jobs from all unassigned projects in its group.
- **specific** - Runs jobs from assigned projects.
- **locked** - Cannot be assigned to other projects.
- **paused** - Not available to run jobs.

**Set up a shared runner manually**

1. Install [GitLab Runner](#) and ensure it's running.
2. Register the runner with this URL:

`https://gitlab.mycluster-tok02-b-889642-d85540d5ac7eb4665421ec4a86b8d368-0000.jp-tok.containers.appdomain.cloud/` URL

And this registration token:

登録トークン

Buttons: Reset registration token, Show Runner installation instructions

## 2.2 OperatorによるGitLab Runnerのインストール – 2. GitLabRunner登録トークン、URLの確認とsecretの作成

### ■ 登録トークンのsecretの作成

#### 1. OpenShiftクラスターにログインする

```
oc login --token=<トークン> --server=<接続先サーバーのURL>
```

#### 2. GitLab Runnerを導入するプロジェクトへ切り替える

```
oc project <プロジェクト名>
```

#### – 実行例

```
$ oc project gitlab-system  
Now using project "gitlab-system" on server "https://c100-e.jp-tok.containers.cloud.ibm.com:32329".
```

## 2.2 OperatorによるGitLab Runnerのインストール -2. GitLabRunner登録トークン、URLの確認とsecretの作成

### ■ 登録トークンのsecretの作成

3.以下のコマンドでGitLab Runnerの登録トークンを利用してsecretのリソースファイルを作成する

```
cat > gitlab-runner-secret.yml << EOF
apiVersion: v1
kind: Secret
metadata:
  name: gitlab-runner-secret
type: Opaque
stringData:
  runner-registration-token: <確認した登録トークンの値>
EOF
```

4.以下のコマンドで、secretを作成する

```
oc apply -f gitlab-runner-secret.yml
```

## 2.2 OperatorによるGitLab Runnerのインストール -2. GitLabRunner登録トークン、URLの確認とsecretの作成

### ■ 登録トークンのsecretの作成

5.以下のコマンドで、secretが作成されたことを確認する

```
oc get secret gitlab-runner-secret
```

- 実行例

```
$ oc get secret gitlab-runner-secret
```

NAME	TYPE	DATA	AGE
gitlab-runner-secret	Opaque	2	40d

## 2.2 OperatorによるGitLab Runnerのインストール – 導入の流れ

導入の流れは以下の通り

1. GitLab Runner Operatorの導入
2. GitLabRunner登録トークン、URLの確認とsecretの作成
3. （自己署名証明書を使用する場合）route証明書の確認とsecretの作成
4. GitLab Runnerの導入
5. 接続確認
6. 補足設定

## 2.2 OperatorによるGitLab Runnerのインストール –3. (自己署名証明書を使用する場合) route証明書の確認とsecretの作成

GitLabでSSL自己署名証明書を使用している場合、Runner側に証明書を登録する必要がある

- GitLabとGitLab Runnerを同一のOpenShift上のプロジェクトにデプロイする場合、Serviceを指定しても疎通はできるがArtifactsアップロードに失敗するため、route経由での接続が必須となる。GitLabがSSL自己署名証明書を使用する場合は、Runnerへの自己署名証明書の登録が必要となる。
- 本ガイドでは、OpenShiftのrouteの証明書を登録する手順を説明する  
参考 : GitLabマニュアル Configure a custom TLS cert  
[https://docs.gitlab.com/runner/configuration/configuring\\_runner\\_openshift.html#configure-a-custom-tls-cert](https://docs.gitlab.com/runner/configuration/configuring_runner_openshift.html#configure-a-custom-tls-cert)

## 2.2 OperatorによるGitLab Runnerのインストール -3. (自己署名証明書を使用する場合) route証明書の確認とsecretの作成

### ■ route証明書の確認

1. OpenShiftコンソールにログインし、左側のメニューから「Networking」を展開し「Routes」をクリックする

The screenshot shows the OpenShift console interface. The left sidebar menu has 'Networking' expanded, with 'Routes' selected. The main content area is titled 'Overview' and 'Cluster'. It is divided into three columns: 'Details', 'Status', and 'Activity'.

**Details:**

- Cluster API Address: `https://c100-e.jp-tok.containers.cloud.ibm.com:32329`
- Cluster ID: `d4a88c6d-5ba0-40fa-ae94-386900ff8a73`
- Provider: IBMCloud
- OpenShift Version: 4.6.22
- Update Channel: Not available

**Status:**

- Cluster: ✔
- Operators: ✔
- Alerts: ⚠ Apr 13, 2:46 pm: Alerts are not configured to be sent to a notification system, meaning that you may not be notified in a timely fashion when important failures occur. Check the OpenShift documentation to learn how to configure notifications with Alertmanager. [Configure](#)

**Activity:**

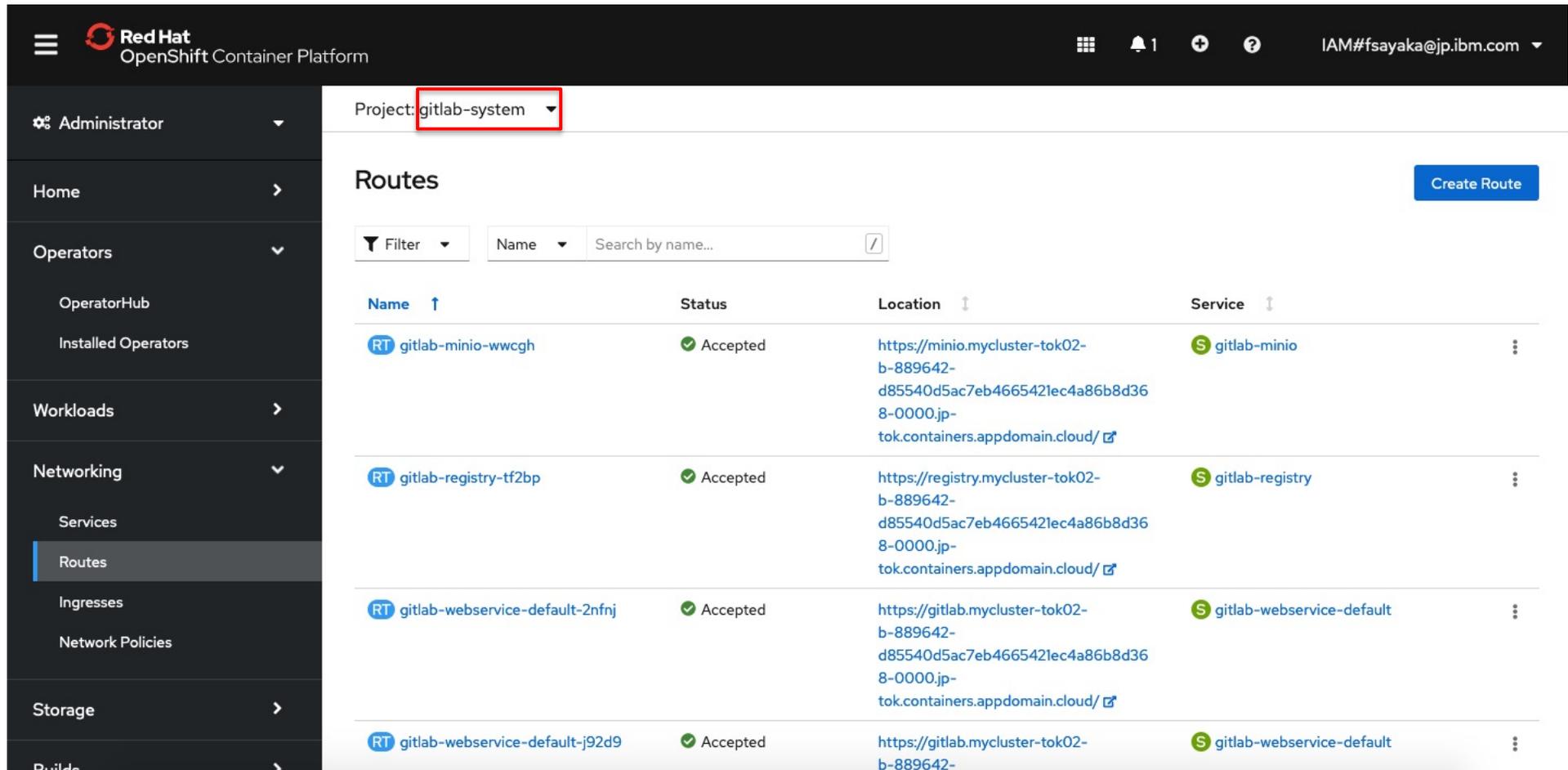
- Ongoing: There are no ongoing activities.
- Recent Events: [Pause](#)
- 17:42 P Stopping contain... [>](#)
- 17:41 P Created containe... [>](#)
- 17:41 P Successfully pulle... [>](#)
- 17:41 P Started container... [>](#)
- 17:41 P Pulling image "reg... [>](#)
- 17:41 P Add eth0 [172.30... [>](#)
- 17:41 P Successfully assign... [>](#)
- 17:29 P Stopping contain... [>](#)

Cluster Utilization (1 Hour):

Resource	Usage	16:45	17:00	17:15	17:30

## 2.2 OperatorによるGitLab Runnerのインストール – 3. (自己署名証明書を使用する場合) route証明書の確認とsecretの作成

- route証明書の確認
2. Projectのリストから「gitlab-system」を選択する



The screenshot shows the Red Hat OpenShift Container Platform administrator console. The top navigation bar includes the Red Hat logo, the text "OpenShift Container Platform", and the user's email address "IAM#fsayaka@jp.ibm.com". The left sidebar contains a menu with categories like Administrator, Home, Operators, Workloads, Networking, and Storage. The "Routes" page is active, showing a list of routes for the "gitlab-system" project. The "gitlab-system" project name is highlighted with a red box. The Routes table has columns for Name, Status, Location, and Service. All listed routes have a status of "Accepted".

Name ↑	Status	Location ↓	Service ↓
RT gitlab-minio-wwcgh	Accepted	https://minio.mycluster-tok02-b-889642-d85540d5ac7eb4665421ec4a86b8d368-0000.jp-tok.containers.appdomain.cloud/	gitlab-minio
RT gitlab-registry-tf2bp	Accepted	https://registry.mycluster-tok02-b-889642-d85540d5ac7eb4665421ec4a86b8d368-0000.jp-tok.containers.appdomain.cloud/	gitlab-registry
RT gitlab-webservice-default-2nfnj	Accepted	https://gitlab.mycluster-tok02-b-889642-d85540d5ac7eb4665421ec4a86b8d368-0000.jp-tok.containers.appdomain.cloud/	gitlab-webservice-default
RT gitlab-webservice-default-j92d9	Accepted	https://gitlab.mycluster-tok02-b-889642-	gitlab-webservice-default

## 2.2 OperatorによるGitLab Runnerのインストール – 3. (自己署名証明書を使用する場合) route証明書の確認とsecretの作成

### ■ route証明書の確認

3. Name欄が「gitlab-web-service-default」で始まり、Locationが「/sidekiq/」で終わらない方のリンクをクリックして開く

The screenshot shows the Red Hat OpenShift Container Platform administrator console. The left sidebar contains navigation menus for Administrator, Home, Operators, Workloads, Networking, and Storage. The main content area displays a table of routes for the project 'gitlab-system'. The table has columns for Name, Status, Location, and Service. The route 'gitlab-webservice-default-2nfnj' is highlighted with a red box.

Name ↑	Status	Location ↓	Service ↓
<a href="#">RT gitlab-minio-wwcgh</a>	Accepted	<a href="https://minio.mycluster-tok02-b-889642-d85540d5ac7eb4665421ec4a86b8d36-8-0000.jp-tok.containers.appdomain.cloud/">https://minio.mycluster-tok02-b-889642-d85540d5ac7eb4665421ec4a86b8d36-8-0000.jp-tok.containers.appdomain.cloud/</a>	<a href="#">S gitlab-minio</a>
<a href="#">RT gitlab-registry-tf2bp</a>	Accepted	<a href="https://registry.mycluster-tok02-b-889642-d85540d5ac7eb4665421ec4a86b8d36-8-0000.jp-tok.containers.appdomain.cloud/">https://registry.mycluster-tok02-b-889642-d85540d5ac7eb4665421ec4a86b8d36-8-0000.jp-tok.containers.appdomain.cloud/</a>	<a href="#">S gitlab-registry</a>
<b><a href="#">RT gitlab-webservice-default-2nfnj</a></b>	Accepted	<a href="https://gitlab.mycluster-tok02-b-889642-d85540d5ac7eb4665421ec4a86b8d36-8-0000.jp-tok.containers.appdomain.cloud/">https://gitlab.mycluster-tok02-b-889642-d85540d5ac7eb4665421ec4a86b8d36-8-0000.jp-tok.containers.appdomain.cloud/</a>	<a href="#">S gitlab-webservice-default</a>
<a href="#">RT gitlab-webservice-default-j92d9</a>	Accepted	<a href="https://gitlab.mycluster-tok02-b-889642-d85540d5ac7eb4665421ec4a86b8d36-8-0000.jp-tok.containers.appdomain.cloud/admin/sidekiq/">https://gitlab.mycluster-tok02-b-889642-d85540d5ac7eb4665421ec4a86b8d36-8-0000.jp-tok.containers.appdomain.cloud/admin/sidekiq/</a>	<a href="#">S gitlab-webservice-default</a>

## 2.2 OperatorによるGitLab Runnerのインストール -3. (自己署名証明書を使用する場合) route証明書の確認とsecretの作成

### ■ route証明書の確認

4. routeの画面でCertificateの値をコピーし、控えておく

The screenshot displays the Red Hat OpenShift Container Platform console interface. The left sidebar shows the navigation menu with 'Routes' selected under the 'Networking' section. The main content area shows the configuration for the 'gitlab-webservice-default' route. Key details include:

- Project:** gitlab-system
- Service:** gitlab-webservice-default
- Target Port:** http-workhorse
- Created At:** Apr 23, 4:50 pm
- Owner:** gitlab-webservice-default

Under the 'TLS Settings' section, the 'Termination Type' is set to 'edge' and 'Insecure Traffic' is set to 'Redirect'. The 'Certificate' section shows a redacted certificate value, with a red box highlighting a copy icon in the bottom right corner of the certificate field.

## 2.2 OperatorによるGitLab Runnerのインストール -3. (自己署名証明書を使用する場合) route証明書の確認とsecretの作成

### ■ route証明書の確認

- secretのリソースファイルを作成する。以下の内容を custom-tls-ca-secret.yaml ファイルとして保存する
  - BEGIN CERTIFICATE以降の行は4. で確認したrouteのCertificateの値を記入する

```
apiVersion: v1
kind: Secret
metadata:
  name: custom-tls-ca
type: Opaque
stringData:
  tls.ca: |
    -----BEGIN CERTIFICATE-----
    MIIeczCCA1ugAwIBAgIBADANBgkqhkiG9w0BAQQFAD..AkGA1UEBhMCR0lx
    .....
    7vQMfXdGsRrXNGRGnX+vWDZ3/zWI0joDtCkNnqEpVn..HoX
    -----END CERTIFICATE-----
```

4. で確認したrouteのCertificateの値を記入

- 以下のコマンドで、secretを作成する

```
oc apply -f custom-tls-ca-secret.yaml
```

## 2.2 OperatorによるGitLab Runnerのインストール -3. (自己署名証明書を使用する場合) route証明書の確認とsecretの作成

### ■ route証明書の確認

7.以下のコマンドで、secretが作成されたことを確認する

```
oc get secret custom-tls-ca
```

– 実行例

```
$ oc get secret custom-tls-ca  
NAME          TYPE    DATA  AGE  
custom-tls-ca Opaque  2      29d
```

## 2.2 OperatorによるGitLab Runnerのインストール – 導入の流れ

導入の流れは以下の通り

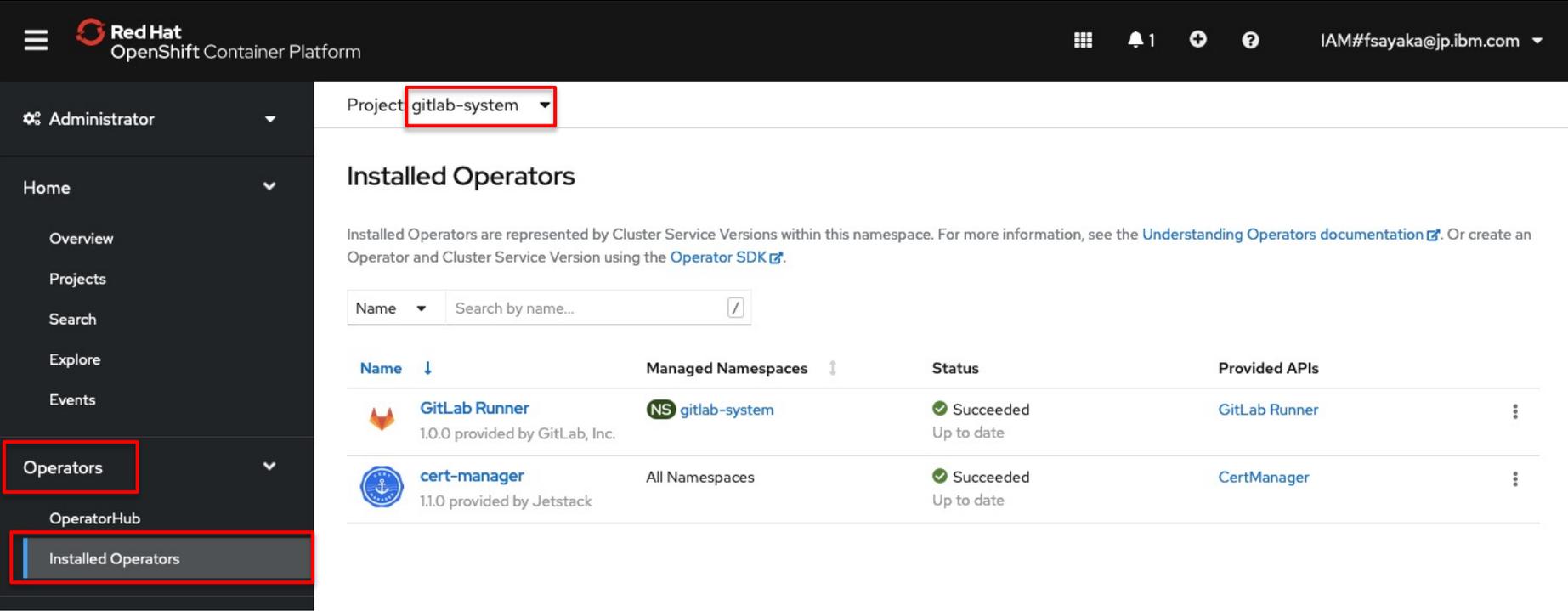
1. GitLab Runner Operatorの導入
2. GitLabRunner登録トークン、URLの確認とsecretの作成
3. (自己署名証明書を使用する場合) route証明書の確認とsecretの作成
4. **GitLab Runnerの導入**
5. 接続確認
6. 補足設定

## 2.2 OperatorによるGitLab Runnerのインストール -4. GitLab Runnerの導入

OpenShiftのWebコンソールを使用して、GitLab Runnerを導入する

1. OpenShiftのWebコンソールに管理者としてログインする

2. 左側のOperators > Installed Operatorsをクリックし、上のProjects:のリストから「gitlab-system」を選択する



Red Hat OpenShift Container Platform

Administrator

Home

- Overview
- Projects
- Search
- Explore
- Events
- Operators**
- OperatorHub
- Installed Operators

Project: **gitlab-system**

### Installed Operators

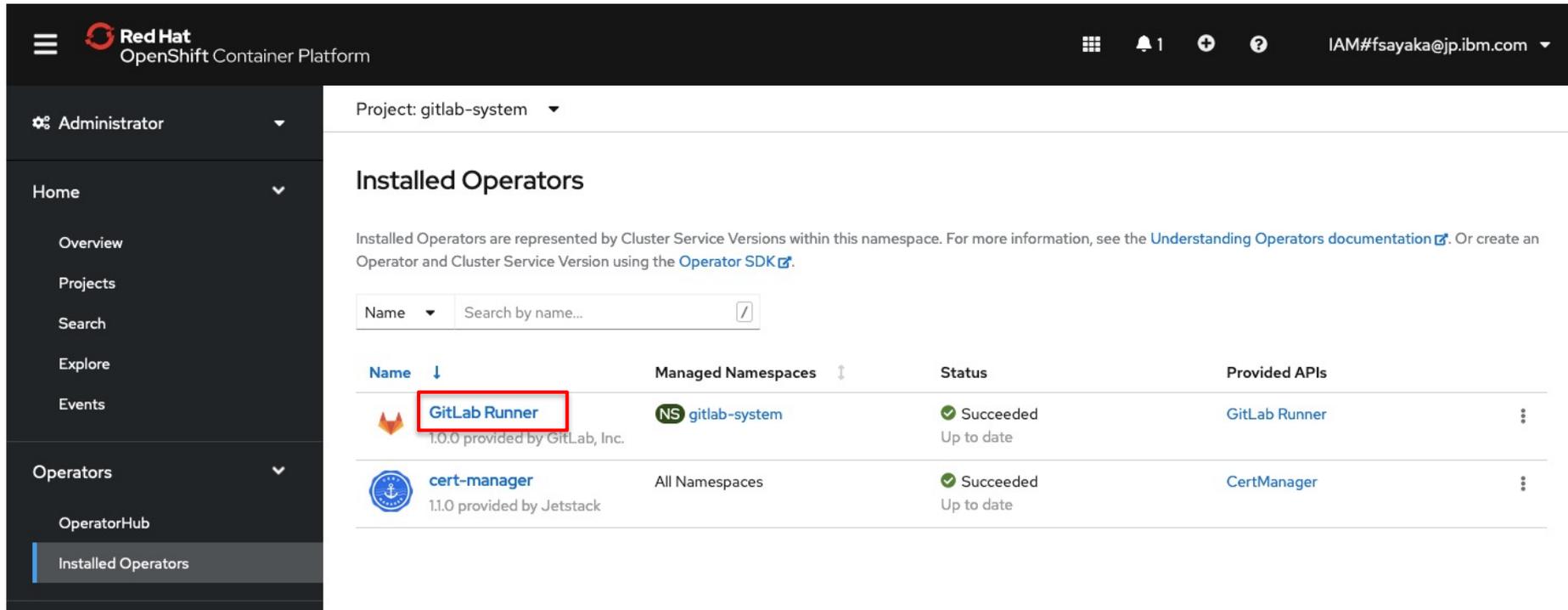
Installed Operators are represented by Cluster Service Versions within this namespace. For more information, see the [Understanding Operators documentation](#). Or create an Operator and Cluster Service Version using the [Operator SDK](#).

Name Search by name...

Name ↓	Managed Namespaces ↑	Status	Provided APIs
 <b>GitLab Runner</b> 1.0.0 provided by GitLab, Inc.	NS gitlab-system	✓ Succeeded Up to date	GitLab Runner
 <b>cert-manager</b> 1.1.0 provided by Jetstack	All Namespaces	✓ Succeeded Up to date	CertManager

## 2.2 OperatorによるGitLab Runnerのインストール -4. GitLab Runnerの導入

### 3.表示された「GitLab Runner」をクリックする



The screenshot shows the Red Hat OpenShift Container Platform interface. The top navigation bar includes the Red Hat logo, the text 'OpenShift Container Platform', and user information 'IAM#fsayaka@jp.ibm.com'. The left sidebar contains navigation options: Administrator, Home, Overview, Projects, Search, Explore, Events, Operators, OperatorHub, and Installed Operators. The main content area is titled 'Installed Operators' and shows a table of installed operators. The 'GitLab Runner' operator is highlighted with a red box.

Project: gitlab-system

### Installed Operators

Installed Operators are represented by Cluster Service Versions within this namespace. For more information, see the [Understanding Operators documentation](#). Or create an Operator and Cluster Service Version using the [Operator SDK](#).

Name Search by name...

Name	Managed Namespaces	Status	Provided APIs
 <b>GitLab Runner</b> 1.0.0 provided by GitLab, Inc.	NS gitlab-system	✓ Succeeded Up to date	<a href="#">GitLab Runner</a>
 <b>cert-manager</b> 1.1.0 provided by Jetstack	All Namespaces	✓ Succeeded Up to date	<a href="#">CertManager</a>

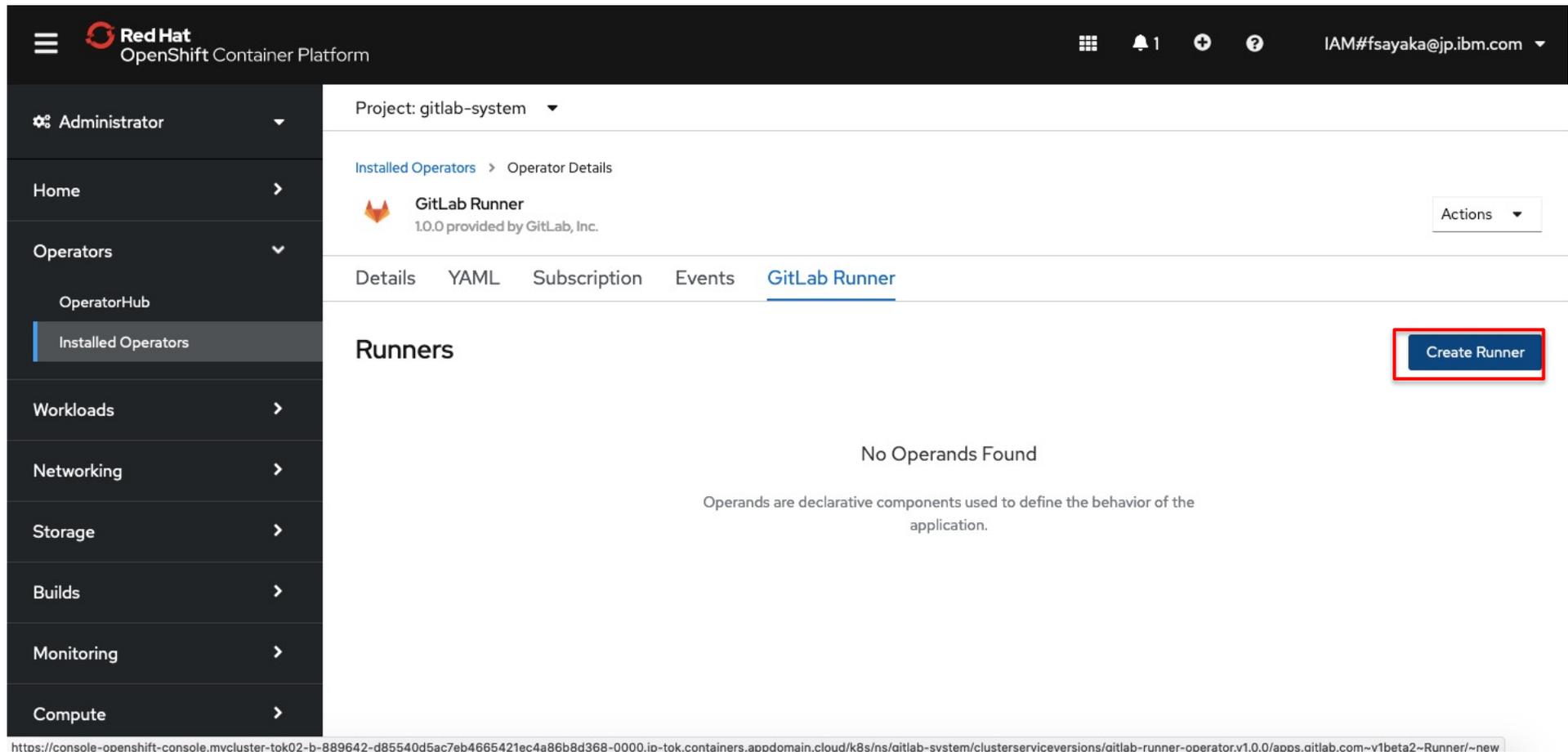
## 2.2 OperatorによるGitLab Runnerのインストール -4. GitLab Runnerの導入

4.表示されたGitLab Runnerの画面で「GitLab Runner」タブをクリックする

The screenshot displays the Red Hat OpenShift Container Platform interface. The top navigation bar includes the Red Hat logo, the text 'OpenShift Container Platform', and user information 'IAM#fsayaka@jp.ibm.com'. The left sidebar shows a menu with 'Administrator', 'Home', 'Operators', 'Workloads', 'Networking', 'Storage', 'Builds', 'Monitoring', and 'Compute'. Under 'Operators', 'Installed Operators' is selected. The main content area shows 'Project: gitlab-system' and 'Installed Operators > Operator Details'. The 'GitLab Runner' operator is listed as '1.0.0 provided by GitLab, Inc.' with an 'Actions' dropdown. Below this, a tabbed interface has 'GitLab Runner' selected and highlighted with a red box. The 'Provided APIs' section shows a 'GitLab Runner' card with a description: 'Runner is the open source project used to run your jobs and send the results back to GitLab' and a 'Create Instance' button. The 'Description' section at the bottom states: 'GitLab Runner is the lightweight, highly-scalable agent that runs your build jobs and sends the results back to a GitLab instance. GitLab Runner works in conjunction with GitLab CI/CD, the open-source continuous integration service included with GitLab CI/CD.' The right sidebar contains metadata: 'Provider: GitLab, Inc.', 'Created At: May 7, 4:11 pm', and 'Links' including 'Gitlab Runner Operator' and 'GitLab Docs'.

## 2.2 OperatorによるGitLab Runnerのインストール -4. GitLab Runnerの導入

5. GitLab Runnerのインスタンスを作成するため「Create Runner」をクリックする



The screenshot shows the Red Hat OpenShift Container Platform console interface. The top navigation bar includes the Red Hat logo, the text "Red Hat OpenShift Container Platform", and user information "IAM#fsayaka@jp.ibm.com". The left sidebar contains a menu with items: Administrator, Home, Operators, OperatorHub, Installed Operators (highlighted), Workloads, Networking, Storage, Builds, Monitoring, and Compute. The main content area shows the "Project: gitlab-system" and "Installed Operators > Operator Details" for "GitLab Runner 1.0.0 provided by GitLab, Inc.". Below this, there are tabs for "Details", "YAML", "Subscription", "Events", and "GitLab Runner". The "Runners" section is currently empty, displaying "No Operands Found" and a message: "Operands are declarative components used to define the behavior of the application." A blue "Create Runner" button is highlighted with a red rectangular box.

https://console-openshift-console.mvcluster-tok02-b-889642-d85540d5ac7eb4665421ec4a86b8d368-0000.io-tok.containers.appdomain.cloud/k8s/ns/aitlab-svstem/clusterserviceversions/aitlab-runner-operator.v1.0.0/apps.aitlab.com~v1beta2~Runner/~new

## 2.2 OperatorによるGitLab Runnerのインストール -4. GitLab Runnerの導入

### 6.表示された画面で以下の値を入力する

- 以下の入力値は[GitLabマニュアル-Install GitLab Runner](#)のyamlファイルを参考に行っている
- 本ガイド作成時は以下以外の設定はデフォルト値のままとした。必要に応じて変更のこと
  - 参考：[GitLabマニュアル - GitLab Runner Operator properties](#)

項目	設定値	備考
Name	gitlab-runner	
GitLab URL	「2.GitLabRunner登録トークン、URLの確認とsecretの作成」で確認したURL	
Registration Token	gitlab-runner-secret	「2.GitLabRunner登録トークン、URLの確認とsecretの作成」で作成したsecret名を設定する
Tags	openshift	
Builder Image	alpine	
Ca	custom-tls-ca-secret	GitLabでSSL自己署名証明書を使用している場合に必要。 「3.（自己署名証明書を使用する場合）route証明書の確認とsecretの作成」で作成したsecret名を設定する

## 2.2 OperatorによるGitLab Runnerのインストール -4. GitLab Runnerの導入

### 6.表示された画面で以下の値を入力する

- 入力画面例

The screenshot shows the 'Create Runner' page in the Red Hat OpenShift Container Platform console. The page is titled 'Create Runner' and includes a sidebar with navigation options like 'OperatorHub', 'Workloads', 'Networking', 'Storage', 'Builds', 'Monitoring', 'Compute', 'User Management', and 'Administration'. The main content area shows the 'Create Runner' form for the 'gitlab-system' project. The form includes a 'Name' field with the value 'gitlab-runner', a 'Labels' field with the value 'app=frontend', a 'GitLab URL' field with the value 'https://gitlab.mycluster-tok02-b-889642-d85540d5ac7eb4665421ec4a86b8d368-0000.jp-t ...', and a 'Registration Token' field with the value 'gitlab-runner-secret'. A note at the top of the form states: 'Note: Some fields may not be represented in this form view. Please select "YAML view" for full control.' The 'GitLab Runner' section on the right provides information about the runner, including the GitLab logo and the text 'GitLab Runner provided by GitLab, Inc. Runner is the open source project used to run your jobs and send the results back to GitLab'.

## 2.2 OperatorによるGitLab Runnerのインストール -4. GitLab Runnerの導入

### 6.表示された画面で以下の値を入力する

- 入力画面例

The screenshot shows the Red Hat OpenShift Container Platform OperatorHub interface. The left sidebar contains navigation options: OperatorHub, Installed Operators, Workloads, Networking, Services, Routes, Ingresses, Network Policies, Storage, Builds, Monitoring, Compute, User Management, and Administration. The main content area displays the configuration for the 'gitlab-system' project. The 'Tags' field is highlighted with a red box and contains the value 'openshift'. Below the 'Tags' field, there is a link to the GitLab documentation: <https://docs.gitlab.com/ee/ci/runners/#use-tags-to-limit-the-number-of-jobs-using-the-runner>. Other configuration options visible include 'Concurrent', 'Check Interval', 'S3', 'Config', 'Cache Shared', and 'Cache Path'.

## 2.2 OperatorによるGitLab Runnerのインストール -4. GitLab Runnerの導入

### 6.表示された画面で以下の値を入力する

- 入力画面例

The screenshot shows the Red Hat OpenShift Container Platform OperatorHub interface. The left sidebar contains navigation options: OperatorHub, Installed Operators, Workloads, Networking, Services, Routes, Ingresses, Network Policies, Storage, Builds, Monitoring, Compute, User Management, and Administration. The main content area displays the configuration for the 'gitlab-system' project. The 'Build Image' field is highlighted with a red box and contains the value 'alpine'. Other fields include Cache Path, Env, Helper Image, Azure, Clone URL, and Image Pull Policy.

Project: gitlab-system

**Cache Path**  
Path defines the Runner Cache path

**Env**  
Accepts configmap name. Provides user mechanism to inject environment variables in the GitLab Runner pod via the key value pairs in the ConfigMap

**Helper Image**  
If specified, overrides the default GitLab Runner helper image

**Build Image**  
The name of the default image to use to run build jobs, when none is specified

**Azure**  
options used to setup Azure blob storage as GitLab Runner Cache

**Clone URL**  
If specified, overrides the default URL used to clone or fetch the Git ref

**Image Pull Policy**  
Always  
ImagePullPolicy sets the Image pull policy. One of Always, Never, IfNotPresent. Defaults to Always if

## 2.2 OperatorによるGitLab Runnerのインストール -4. GitLab Runnerの導入

### 6.表示された画面で以下の値を入力する

- 入力画面例

The screenshot shows the Red Hat OpenShift Container Platform console interface. The left sidebar contains navigation menus for OperatorHub, Workloads, Networking, Storage, Builds, Monitoring, Compute, User Management, and Administration. The main content area displays the configuration page for the 'gitlab-system' project. The 'Ca' field, which is highlighted with a red box, contains the value 'custom-tls-ca-secret'. Below the 'Ca' field, there are 'Create' and 'Cancel' buttons.

Project: gitlab-system

If specified, overrides the default URL used to clone or fetch the Git ref

**Image Pull Policy**

Always

ImagePullPolicy sets the Image pull policy. One of Always, Never, IfNotPresent. Defaults to Always if :latest tag is specified, or IfNotPresent otherwise. More info: <https://kubernetes.io/docs/concepts/containers/images#updating-images>

**Gcs**

options used to setup GCS (Google Container Storage) as GitLab Runner Cache

**Serviceaccount**

allow user to override service account used by GitLab Runner

**Cache Type**

Type of cache used for Runner artifacts Options are: gcs, s3, azure

**Ca**

custom-tls-ca-secret

Name of tls secret containing the custom certificate authority (CA) certificates

Create Cancel

## 2.2 OperatorによるGitLab Runnerのインストール -4. GitLab Runnerの導入

7.入力が終わったら、「Create」をクリックする

The screenshot shows the Red Hat OpenShift Container Platform console interface. The left sidebar contains navigation options: OperatorHub, Installed Operators, Workloads, Networking, Services, Routes, Ingresses, Network Policies, Storage, Builds, Monitoring, Compute, User Management, and Administration. The main content area displays the configuration for the 'gitlab-system' project. The configuration includes fields for 'Image Pull Policy' (set to 'Always'), 'Gcs' (options for GCS as GitLab Runner Cache), 'Serviceaccount', 'Cache Type', and 'Ca' (set to 'custom-tls-ca-secret'). At the bottom, there are two buttons: 'Create' (highlighted with a red box) and 'Cancel'.

## 2.2 OperatorによるGitLab Runnerのインストール -4. GitLab Runnerの導入

8.しばらく待ち、以下のようにStatusが Phase: Running になることを確認する

The screenshot shows the Red Hat OpenShift Container Platform interface. The top navigation bar includes the Red Hat logo, the text 'Red Hat OpenShift Container Platform', and user information 'IAM#fsayaka@jp.ibm.com'. The left sidebar contains navigation options: Administrator, Home, Operators (with sub-items OperatorHub and Installed Operators), and Workloads (with sub-items Pods, Deployments, and Deployment Configs). The main content area shows the 'Project: gitlab-system' and 'Installed Operators > Operator Details' for 'GitLab Runner' (version 1.0.0). Below this, there are tabs for 'Details', 'YAML', 'Subscription', 'Events', and 'GitLab Runner'. The 'Runners' section features a 'Create Runner' button and a search bar. A table lists the runners:

Name	Kind	Status	Labels
gitlab-runner	Runner	Phase: <span style="color: green;">✔</span> Running	No labels

## 2.2 OperatorによるGitLab Runnerのインストール -4. GitLab Runnerの導入

9.以下のコマンドを実行して、GitLab Runnerがインストールされていることを確認する（事前に oc loginにてOpenShiftクラスターへログインを実施のこと）

```
oc get runners -n gitlab-system
```

– 実行例

```
$ oc get runners  
NAME          AGE  
gitlab-runner 105s
```

## 2.2 OperatorによるGitLab Runnerのインストール -4. GitLab Runnerの導入

10.以下のコマンドを実行して、Runnerのpodが表示され、STATUSが「Running」であることを確認する

```
oc get pod -l app.kubernetes.io/name=gitlab-runner -n gitlab-system
```

- 実行例

```
$ oc get pod -l app.kubernetes.io/name=gitlab-runner -n gitlab-system
NAME                                READY STATUS  RESTARTS  AGE
gitlab-runner-runner-66456cb954-nhsh9 1/1   Running  0         2m36s
```

## 2.2 OperatorによるGitLab Runnerのインストール – 導入の流れ

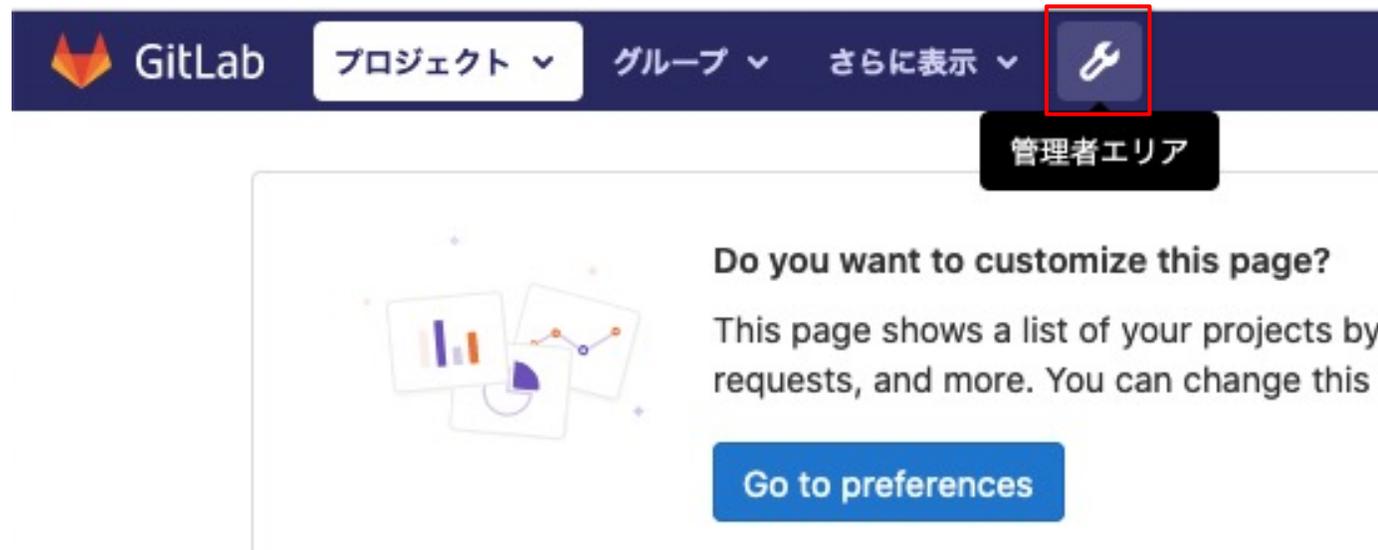
導入の流れは以下の通り

1. GitLab Runner Operatorの導入
2. GitLabRunner登録トークン、URLの確認とsecretの作成
3. (自己署名証明書を使用する場合) route証明書の確認とsecretの作成
4. GitLab Runnerの導入
- 5. 接続確認**
6. 補足設定

## 2.2 OperatorによるGitLab Runnerのインストール -5. 接続確認

GitLabの画面から、GitLab Runnerが接続したことを確認する  
以降は、共有Runnerとして登録した場合を例に説明する

1. 管理者としてGitLabにログインし、管理者エリアアイコンをクリックする



## 2.2 OperatorによるGitLab Runnerのインストール -5. 接続確認

### 2. 「概要」 の下の 「Runner」 をクリックする

The screenshot shows the GitLab Admin Area interface. The top navigation bar includes the GitLab logo, navigation menus for 'プロジェクト', 'グループ', and 'さらに表示', a search bar, and utility icons. The left sidebar contains the '管理者エリア' (Admin Area) menu with options like 'ダッシュボード', 'プロジェクト', 'ユーザー', 'グループ', 'ジョブ', 'Runner', and 'Gitaly サーバー'. The 'Runner' option is highlighted with a red box. The main content area is titled '管理者エリア > Runner' and contains the following information:

Runners are processes that pick up and execute CI/CD jobs for GitLab. You can register runners as separate users, on separate servers, and on your local machine. Register as many runners as you want.

Runners can be:

- **shared** - Runs jobs from all unassigned projects.
- **group** - Runs jobs from all unassigned projects in its group.
- **specific** - Runs jobs from assigned projects.
- **locked** - Cannot be assigned to other projects.
- **paused** - Not available to run jobs.

**Set up a shared runner manually**

1. [Install GitLab Runner and ensure it's running.](#)
2. Register the runner with this URL:  
`https://gitlab.mycluster-tok02-b-889642-d85540d5ac7eb4665421ec4a86b8d368-0000.jp-tok.containers.appdomain.cloud/`

And this registration token:  
`MTVswEJluTpkliIHD9ntrnPKRpBd4qwmGpYdonnb23VDZATxt1kvHkbaQxKQd2`

Buttons: [Reset registration token](#), [Show Runner installation instructions](#)

## 2.2 OperatorによるGitLab Runnerのインストール -5. 接続確認

- 右側の一覧に接続しているRunnerが表示される
- Description欄にはRunnerのPod名が表示される

Admin Area > Runners

Runners are processes that pick up and execute CI/CD jobs for GitLab. You can register runners as separate users, on separate servers, and on your local machine. Register as many runners as you want.

Runners can be:

- shared** - Runs jobs from all unassigned projects.
- group** - Runs jobs from all unassigned projects in its group.
- specific** - Runs jobs from assigned projects.
- locked** - Cannot be assigned to other projects.
- paused** - Not available to run jobs.

**Set up a shared runner manually**

- Install GitLab Runner and ensure it's running.
- Register the runner with this URL:  
`https://gitlab.mycluster-tok02-b-889642-d85540d5ac7eb4665421ec4a86b8d368-0000.jp-tok.containers.appdomain.cloud/`

And this registration token:  
`MTVswEJluTpkliIHD9ntrnPKRpBd4qwmTgPYdonnb23VDZATxt1kvHkbAqXKQd2`

Reset registration token

Show Runner installation instructions

Recent searches Search or filter results... Created date Runners currently online: 1

Type/State	Runner token	Description	Version	IP Address	Projects	Jobs	Tags	Last contact
shared	j5Epnbh8	gitlab-runner-runner-66456...	13.10.0	172.30.64.135	n/a	0	openshift	just now

## 2.2 OperatorによるGitLab Runnerのインストール – 導入の流れ

導入の流れは以下の通り

1. GitLab Runner Operatorの導入
2. GitLabRunner登録トークン、URLの確認とsecretの作成
3. (自己署名証明書を使用する場合) route証明書の確認とsecretの作成
4. GitLab Runnerの導入
5. 接続確認
6. **補足設定**

## 2.2 OperatorによるGitLab Runnerのインストール –6.補足設定

### ■ タグなしのジョブからも実行可能とする設定

Runner作成時にTagsで指定したタグ名を指定しないジョブも実行可能にする設定を説明する

#### 1.設定対象のRunnerのトークンのリンクをクリックする

The screenshot shows the GitLab Admin Area for Runners. The left sidebar contains navigation options like Overview, Dashboard, Projects, Users, Groups, Jobs, Runners, and Gitaly Servers. The main content area provides information about Runners and instructions for manual setup. A table at the bottom lists registered runners with columns for Type/State, Runner token, Description, Version, IP Address, Projects, Jobs, Tags, and Last contact. The runner token 'j5Epnbh8' is highlighted with a red box.

**Runners**

Runners are processes that pick up and execute CI/CD jobs for GitLab. You can register runners as separate users, on separate servers, and on your local machine. Register as many runners as you want.

Runners can be:

- shared** - Runs jobs from all unassigned projects.
- group** - Runs jobs from all unassigned projects in its group.
- specific** - Runs jobs from assigned projects.
- locked** - Cannot be assigned to other projects.
- paused** - Not available to run jobs.

**Set up a shared runner manually**

1. Install GitLab Runner and ensure it's running.
2. Register the runner with this URL:  
`https://gitlab.mycluster-tok02-b-889642-d85540d5ac7eb4665421ec4a86b8d368-0000.jp-tok.containers.appdomain.cloud/`

And this registration token:  
`MTVswEJluTpkliILHD9nt rnPKRpBd4qwmTgPYdonnb23VDZATxt1kvHkbAqXKQd2`

Buttons: Reset registration token, Show Runner installation instructions

Runners currently online: 1

Type/State	Runner token	Description	Version	IP Address	Projects	Jobs	Tags	Last contact
shared	<a href="#">j5Epnbh8</a>	gitlab-runner-runner-66456...	13.10.0	172.30.64.135	n/a	0	openshift	just now

## 2.2 OperatorによるGitLab Runnerのインストール –6.補足設定

### ■ タグなしのジョブからも実行可能とする設定

2.Runnerの設定画面で「タグのないジョブの実行」にチェックし、「変更を保存」をクリックする

The screenshot shows the GitLab Runner configuration interface. At the top, there is a navigation bar with the GitLab logo and various menu items. Below the navigation bar, there is a sidebar on the left with a menu for '管理者エリア' (Admin Area) containing options like '概要' (Overview), 'ダッシュボード' (Dashboard), 'プロジェクト' (Projects), 'ユーザー' (Users), 'グループ' (Groups), 'ジョブ' (Jobs), 'Runner', and 'Gitaly サーバー' (Gitaly Servers). The main content area is titled 'This runner processes jobs for all unassigned projects.' and contains several settings:

- 有効** (Enabled):  Paused runners don't accept new jobs
- 保護されています** (Protected):  この Runner は保護ブランチ上で起動されたパイプラインでのみ実行できます。
- タグのないジョブの実行** (Run jobs without tags):  このランナーがタグのないジョブを選択できるかどうかを示します。
- 現在のプロジェクトをロックする** (Lock current project):  Runner がロックされている場合、他のプロジェクトに割り当てることができません
- IP アドレス** (IP address): 172.30.253.36
- 説明** (Description): gitlab-runner-runner-8554c7c895-z4gvw
- ジョブタイムアウトの最大値** (Maximum job timeout):  Enter the number of seconds, or other human-readable input, like "1 hour". This timeout takes precedence over lower timeouts set for the project.
- タグ** (Tags): openshift  
You can set up jobs to only use runners with specific tags. Separate tags with commas.

At the bottom of the page, there is a button labeled '変更を保存' (Save changes), which is highlighted with a red box.

# インストールの流れ

- 以下の手順でインストールを行う

**1**

OperatorによるGitLabのインストール

**2**

OperatorによるGitLab Runnerのインストール

**3**

ライセンスの設定

## 2.3 ライセンスの設定

- GitLabの有償版であるGitLab Ultimateを使用するにはライセンスの設定が必要になる

参考 : GitLab マニュアル - 料金

<https://www.gitlab.jp/pricing/#self-managed>

- 本ガイドでは、以下を参考にしたライセンス・キー取得後の設定手順を説明する

参考 : GitLab マニュアル - Uploading your license

[https://docs.gitlab.com/ee/user/admin\\_area/license.html#uploading-your-license](https://docs.gitlab.com/ee/user/admin_area/license.html#uploading-your-license)

- ライセンスの取得については別途営業担当者を確認のこと
- GitLabが提供する評価ライセンスについては以下のリンク先を参照のこと

参考 : GitLab マニュアル - Free trial ページ

<https://about.gitlab.com/free-trial/self-managed/>

## 2.3 ライセンスの設定

---

導入の流れは以下の通り

1. ライセンス・キーの確認
2. ライセンスの設定

## 2.3 ライセンスの設定 -1.ライセンス・キーの確認

導入の流れは以下の通り

1. ライセンス・キーの確認
2. ライセンスの設定

## 2.3 ライセンスの設定 -1.ライセンス・キーの確認

製品購入後または評価ライセンスの申請後、取得したライセンス・キーを確認する

評価ライセンスの場合、以下のようなメールを受信する

### [EXTERNAL] GitLab Self-managed Trial License Key for fsayaka@jp.ibm.com

GitLab to me  
Please respond to leads@gitlab.com

11:23 AM  
[Show more](#)

#### This Message Is From an External Sender

This message came from outside your organization.

curved top shadow

GitLab

Your trial license key for GitLab Self-managed is at the end of this message.  
Get started today:

- [Download GitLab Enterprise Edition](#)
- [Install your license key](#)
- Watch the [GitLab Overview \(18min demo\)](#)
- [Review the documentation](#)

Please reply if you need any help. Trial key for fsayaka@jp.ibm.com:

ライセンス・キー

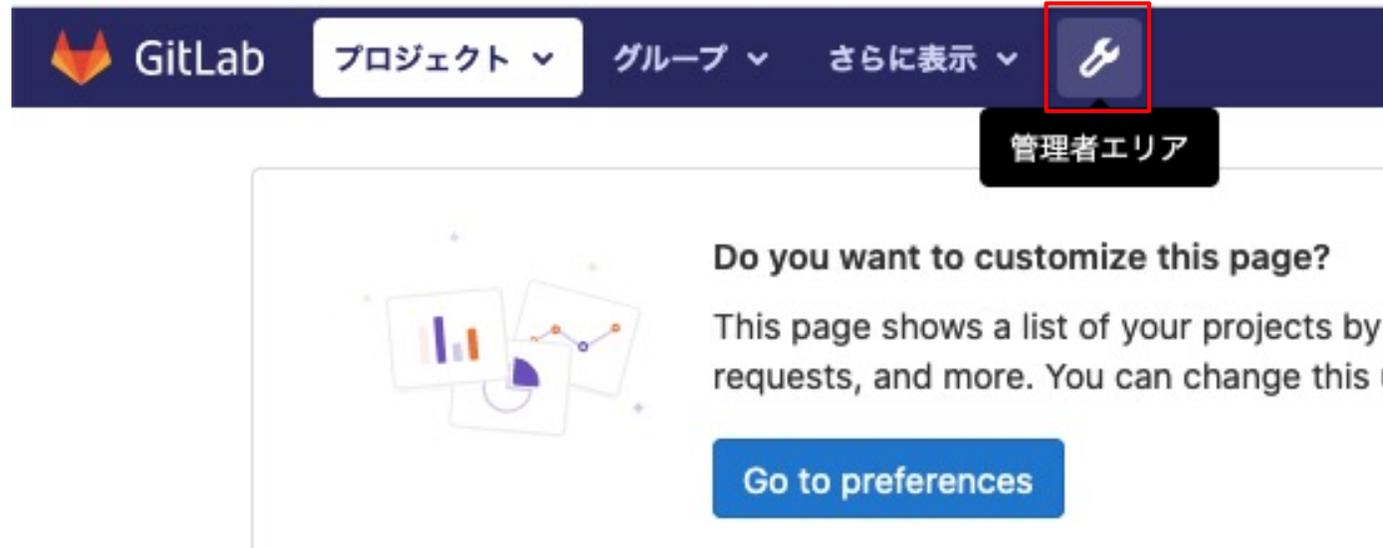
## 2.3 ライセンスの設定 -2.ライセンスの設定

導入の流れは以下の通り

1. ライセンス・キーの確認
2. ライセンスの設定

## 2.3 ライセンスの設定 -2.ライセンスの設定

1. GitLabに管理者としてログインし、管理者エリアアイコンをクリックする



## 2.3 ライセンスの設定 -2.ライセンスの設定

2.左側の「ライセンス」をクリックし、「Upload New License」をクリックする

The screenshot shows the GitLab Admin Area interface. The left sidebar contains a navigation menu with the following items: 管理者エリア, 概要, 分析, 監視, メッセージ, システムフック, アプリケーション, 不正利用レポート (0), **ライセンス** (highlighted with a red box), Kubernetes, Geo, デプロイキー, サービス テンプレート, and サイドバーを隠す. The main content area is titled 'あなたのライセンス' and features two buttons: 'Upload New License' (highlighted with a red box) and 'Buy License'. Below the buttons, there is a central illustration of a toolbox with a padlock, surrounded by icons representing a lock, an eye, a refresh cycle, and a share icon. The text below the illustration reads: 'ライセンスがありません。 GitLab Ultimateの無料トライアルは、義務や支払いの詳細なしで開始できます。' and a 'Start free trial' button is positioned at the bottom center. The URL at the bottom of the page is: https://gitlab.mycluster-tok02-b-889642-d85540d5ac7eb4665421ec4a86b8d368-0000.jp-tok.containers.appdomain.cloud/admin/license/new

## 2.3 ライセンスの設定 -2.ライセンスの設定

### 3. 「Upload License」画面が表示される

ライセンスの提供形態に合わせて以下のいずれかを選択する

- .gitlab-license ファイルとして提供された場合、「Upload .gitlab-license file」を選択する
- ライセンス・キーの値が提供された場合、「Enter license key」を選択する

GitLab プロジェクト ▾ グループ ▾ さらに表示 ▾ 🔗

管理者エリア > Upload License

### Upload License

To start using GitLab Enterprise Edition, upload the `.gitlab-license` file or enter the license key you have received from GitLab Inc.

Upload `.gitlab-license` file  
 Enter license key

License file  ファイルが選択されていません。

Unless otherwise agreed to in writing with GitLab, by clicking "Upload License" you agree that your use of GitLab Software is subject to the [Terms of Service](#).

## 2.3 ライセンスの設定 -2.ライセンスの設定

4. 3.で「Enter license key」を選択した場合「License Key」欄にキーを入力する
- 3.で「Upload .gitlab-license file」を選択した場合は、License file欄の「選択」ボタンを押してファイルを選択する
- 「Enter license key」を選択した場合の入力例

GitLab プロジェクト ▾ グループ ▾ さらに表示 ▾

管理者エリア > Upload License

### Upload License

To start using GitLab Enterprise Edition, upload the `.gitlab-license` file or enter the license key you have received from GitLab Inc.

Upload `.gitlab-license` file

Enter license key

License key

ライセンス・キー

Unless otherwise agreed to in writing with GitLab, by clicking "Upload License" you agree that your use of GitLab Software is subject to the [Terms of Service](#).

Upload License

## 2.3 ライセンスの設定 -2.ライセンスの設定

5. 「Terms of Service」のチェックボックスをチェックして、「Upload License」をクリックする

GitLab プロジェクト ▾ グループ ▾ さらに表示 ▾ 🔗

管理者エリア > Upload License

### Upload License

To start using GitLab Enterprise Edition, upload the `.gitlab-license` file or enter the license key you have received from GitLab Inc.

Upload `.gitlab-license` file  
 Enter license key

License key

ライセンス・キー

Unless otherwise agreed to in writing with GitLab, by clicking "Upload License" you agree that your use of GitLab Software is subject to the [Terms of Service](#).

Upload License

## 2.3 ライセンスの設定 -2.ライセンスの設定

6. ライセンスのアップロードが完了するとライセンス情報が表示される  
Planに「Ultimate」と表示されればUltimateのライセンスが設定されている

GitLab プロジェクト グループ さらに表示

管理者エリア > ライセンス

ライセンスは正常にアップロードされ、アクティブになりました。あなたは、詳細が見えます。

**Your License** [Upload New License](#) [Buy License](#)

Licensed to

Name: **Sayaka Fukushima**

Email: **fsayaka@jp.ibm.com**

Company: **IBM Sytems Engineering Co.Ltd**

詳細

Plan: **Ultimate**

Uploaded: **たった今**

Started: **2 時間前**

有効期限: **Free trial will expire in 30 day**

License ID: **177786**

**ライセンスをダウンロード**

Your license will be included in your GitLab backup and will survive upgrades, so in normal usage you should never need to re-upload your `.gitlab-license` file.

それでも、どこかにバックアップを取っておくことをお勧めします。そうでなければ、もし紛失してしまって必要になった場合は、GitLab Inc. に再送信を依頼する必要があります。

[ライセンスをダウンロード](#) [Customer Portal](#)

**Remove license**

If you remove this license, GitLab will fall back on the previous license, if any.

以前のライセンスがない場合や、以前のライセンスの有効期限が切れている場合は、新しい有効なライセンスがアップロードされるまで GitLab の一部の機能がブロックされます。

[Remove license](#)

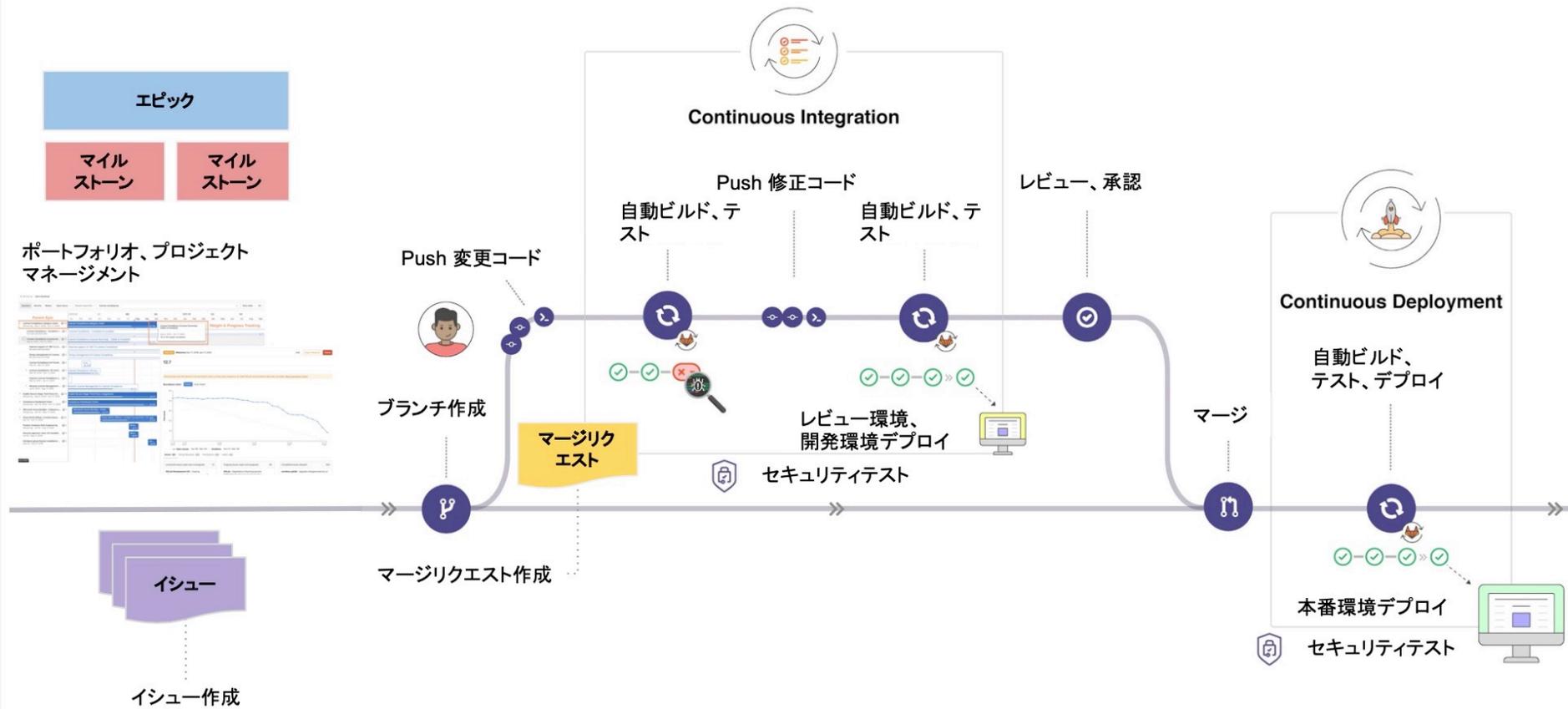
管理者エリア

- 概要
- 分析
- 監視
- メッセージ
- システムフック
- アプリケーション
- 不正利用レポート 0
- ライセンス
- Kubernetes
- ブッシュルール
- Geo
- デプロイキー
- サイドバーを隠す

- 3.1. GitLab CI/CD とは

## 3. GitLab CI/CDの利用

# GitLab: DevOpsベストプラクティスの組み込み



5

GitLab合同会社「GitLabのマージリクエストとCI/CDを活用した並行開発」より抜粋

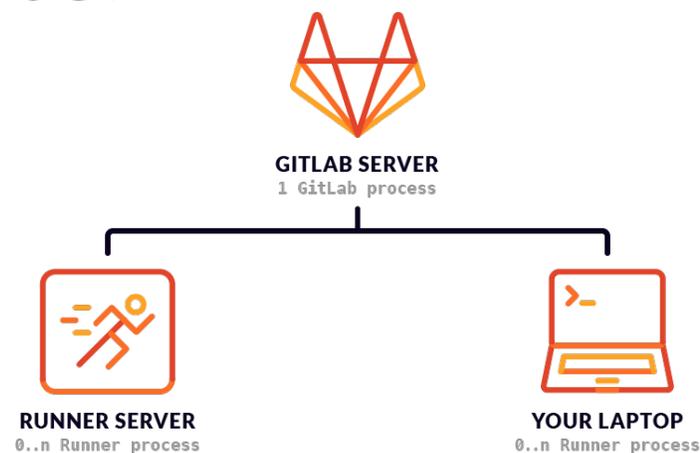
# GitLab CI/CD

## ■ GitLab CI/CD は…

- GitLab の一部であり、その状態をデータベースに保存する API を持つ Web アプリケーション。
- プロジェクト/ビルドを管理するためのユーザーインターフェースを提供する。
- リポジトリのルートに配置された.gitlab-ci.ymlというファイルによって構成される。

## ■ GitLab Runner は…

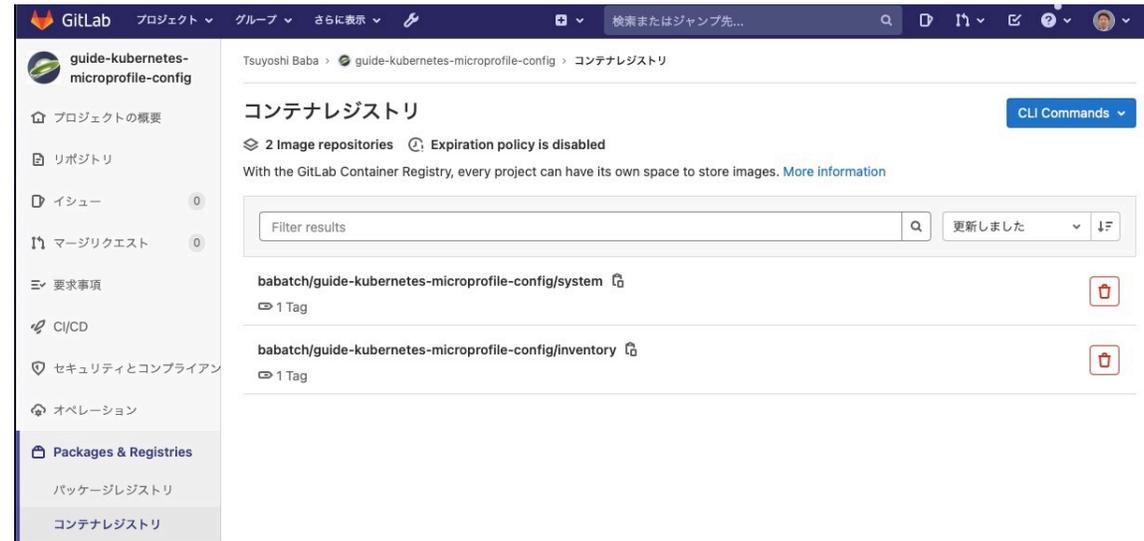
- ビルドを実行するアプリケーション。
- GitLab とは別にインストールする必要がある。
- インストール後、GitLab サーバーに登録すると使用可能になる。



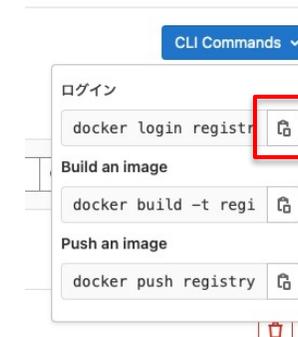
<https://www.gitlab.jp/stages-devops-lifecycle/continuous-integration/>

# GitLab コンテナレジストリ

- 各 GitLab プロジェクトが持つことのできる Docker コンテナレジストリ。
  - Openshift 環境に導入する場合でも、Openshift の内部レジストリとは別物。



- CLI Commands からレジストリ名をクリップボードに入手可能
  - 例) ログイン
    - `docker login <レジストリ名>`

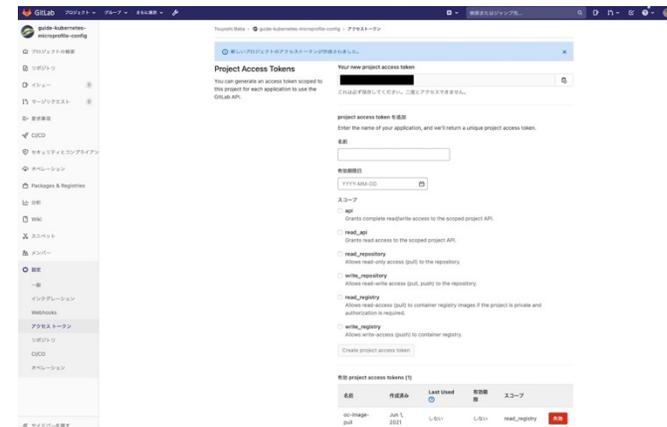


# GitLab コンテナレジストリのイメージプルシークレット

- Kubernetes で GitLab コンテナレジストリから pull する場合は、イメージプルシークレットが必要。
  - "Creating a project access token" に従って Project access token を作成する。
    - [https://docs.gitlab.com/ee/user/project/settings/project\\_access\\_tokens.html#creating-a-project-access-token](https://docs.gitlab.com/ee/user/project/settings/project_access_tokens.html#creating-a-project-access-token)
    - read\_repository をスコープに含める。
    - 指定した「名前」と生成された「アクセストークン」を保存しておく。
  - デプロイ先の Openshift プロジェクトへ移動する。
- 以下のコマンドで imagePullSecret を作成する。

```
oc <プロジェクト名>
```

```
oc create secret docker-registry <Secret名> --docker-server=<レジストリ名> --docker-username=<Project access token名> --docker-password=<Project access token> --docker-email=<eメール・アドレス(適当でよい)>
```



Project access token の生成

- 3.2. GitLab CI/CD の用語

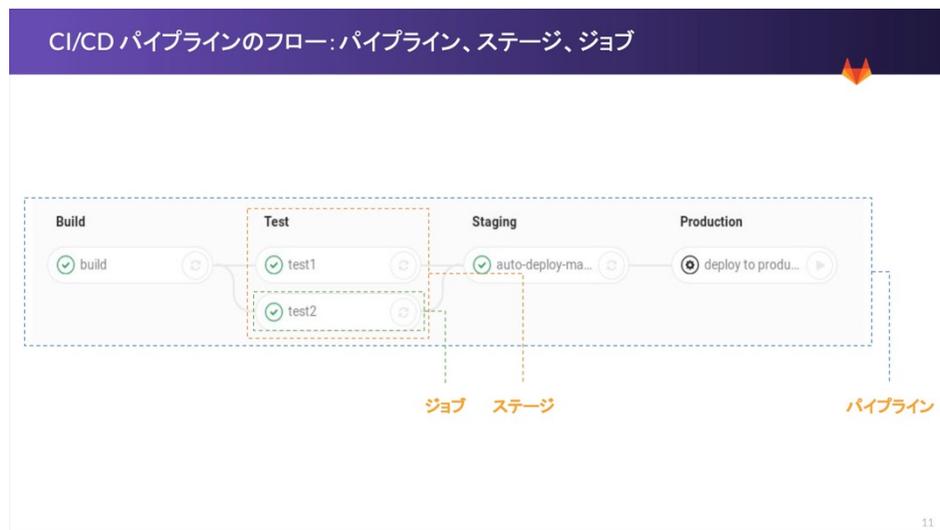
# 3. GitLab CI/CDの利用

# パイプライン、ステージ、ジョブと .gitlab-ci.yml

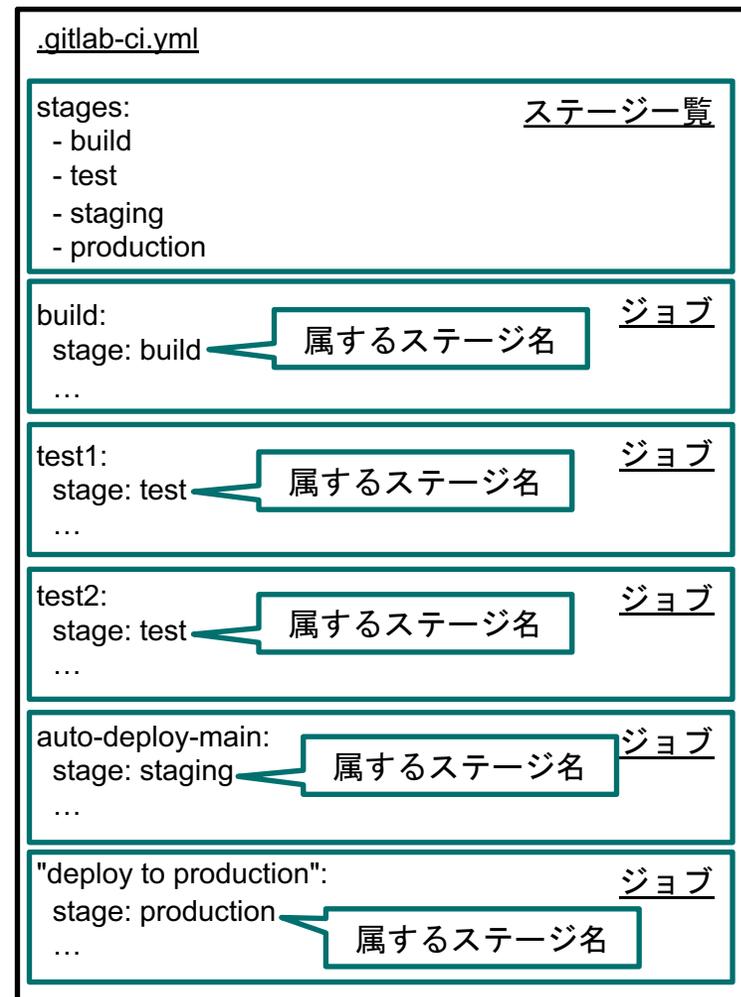
## ■ より複雑なパイプラインについてや、パイプラインについてより詳しくは…

### – CI/CD Pipeline

- <https://docs.gitlab.com/ee/ci/pipelines/>



GitLab合同会社「GitLabのマージリクエストとCI/CDを活用した並行開発」より抜粋



# キャッシュとアーティファクト

	キャッシュ	アーティファクト
ユースケース	依存を解決するためにダウンロードしたパッケージなどを、後続のジョブで使い回すために使用する。	ステージ間でビルドの中間生成物を受け渡すために使用する。
定義する場所	ジョブに <code>cache:</code> キーワードを用いて定義する。	ジョブに <code>artifacts:</code> キーワードを用いて定義する。
利用できる場所	後続のパイプライン 同一の依存関係 ( <code>dependencies:</code> ) を持つ場合、同じパイプライン内の後続のジョブ単位	同じパイプライン内の後続のステージ内のジョブ
共有範囲	異なるプロジェクトはキャッシュを共有できない。	異なるプロジェクトはキャッシュを共有できない。
保管場所	GitLab Runner	GitLab サーバー

# 変数

- 以下の目的で CI/CD 変数を使用することができる。
  - ジョブやパイプラインの振る舞いを制御する。
  - 再利用したい値を保管する。
  - .gitlab-ci.yml に値をハードコードすることを避ける。
  
- 以下の変数を使用することができる。
  - あらかじめ定義されている変数 (predefined CI/CD variables)
  - プロジェクト CI/CD 変数
  - グループ CI/CD 変数
  - インスタンス CI/CD 変数

# あらかじめ定義されている変数

- すべての GitLab CI/CD パイプラインで使用することができる。
- GitLab Runner の新しいバージョンでは使用できる変数が増えていることがある。
- 以下のジョブで内容を確認することができる。

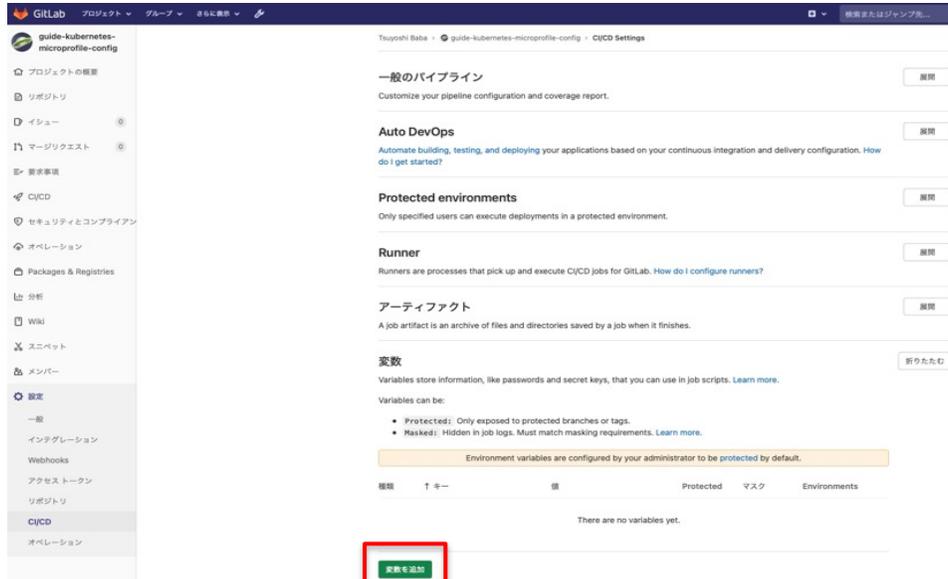
```
job_name:  
  script:  
    - export
```

- 値を確認した例

```
export CI='true'  
export CI_API_V4_URL='https://gitlab.mycluster-tok02-b-889642-d85540d5ac7eb4665421ec4a86b8d368-0000.jp-tok.containers.appdomain.cloud/api/v4/'  
export CI_BUILDS_DIR='/builds'  
export CI_BUILD_BEFORE_SHA='1b96a1dd52dda223580712d96af8a585cb85e3fd'  
export CI_BUILD_ID='234'  
export CI_BUILD_NAME='build'  
export CI_BUILD_REF='4fa86ac83759ff58f3cccc3147a02b10b1836ac0'  
export CI_BUILD_REF_NAME='master'  
export CI_BUILD_REF_SLUG='master'  
export CI_BUILD_STAGE='build'  
export CI_BUILD_TOKEN='[MASKED]'  
export CI_COMMIT_BEFORE_SHA='1b96a1dd52dda223580712d96af8a585cb85e3fd'  
export CI_COMMIT_BRANCH='master'  
export CI_COMMIT_DESCRIPTION=""  
export CI_COMMIT_MESSAGE='add GIT_SSL_NO_VERIFY to .gitlab-ci.yml.'  
export CI_COMMIT_REF_NAME='master'  
export CI_COMMIT_REF_PROTECTED='true'  
export CI_COMMIT_REF_SLUG='master'  
export CI_COMMIT_SHA='4fa86ac83759ff58f3cccc3147a02b10b1836ac0'  
export CI_COMMIT_SHORT_SHA='4fa86ac8'  
export CI_COMMIT_TIMESTAMP='2021-05-27T16:23:33+09:00'  
export CI_COMMIT_TITLE='add GIT_SSL_NO_VERIFY to .gitlab-ci.yml.'  
export CI_CONCURRENT_ID='0'  
export CI_CONCURRENT_PROJECT_ID='0'  
export CI_CONFIG_PATH='.gitlab-ci.yml'  
export CI_DEFAULT_BRANCH='master'  
export CI_DISPOSABLE_ENVIRONMENT='true'  
...
```

# プロジェクト CI/CD 変数

- プロジェクト内で使用可能な CI/CD 変数
- プロジェクト設定 → CI/CD → 変数 から設定することができる。



## 変数

Variables store information, like passwords and secret keys, that you can use in job scripts. [Learn more.](#)

Variables can be:

- **Protected:** Only exposed to protected branches or tags.
- **Masked:** Hidden in job logs. Must match masking requirements. [Learn more.](#)

Environment variables are configured by your administrator to be **protected** by default.

種類	↑ キー	値	Protected	マスク	Environments
Variable	SampleProjectVariable	*****	✓	×	全て(デフォ...

値を表示する

変数を追加

折りたたむ

# グループ CI/CD 変数とインスタンス CI/CD 変数

- プロジェクト CI/CD 変数とほぼ同様に設定することができる。
  - グループ CI/CD 変数:      グループの設定 → CI/CD → 変数
  - インスタンス CI/CD 変数: (管理者エリアで) 設定 → CI/CD → 変数

The image displays two screenshots of the GitLab web interface, illustrating the configuration of CI/CD variables at different levels.

**Left Screenshot: Group CI/CD Settings**  
 This screenshot shows the 'CI/CD Settings' page for a group named 'gitlab-git'. The 'Variables' section is expanded, showing a table with columns for 'Name', 'Value', 'Protected', and 'Masked'. A message states: "Environment variables are configured by your administrator to". Below the table, there is a green button labeled "変数を追加" (Add variable). The left sidebar shows the navigation menu with '設定' (Settings) selected.

**Right Screenshot: Instance CI/CD Settings**  
 This screenshot shows the 'CI/CD Settings' page in the 'Admin Area' (管理者エリア). The 'Variables' section is expanded, showing a table with columns for 'Name', 'Value', 'Protected', and 'Masked'. A message states: "Environment variables on this GitLab instance are configured to be protected by default." Below the table, there is a green button labeled "変数を追加" (Add variable). The left sidebar shows the navigation menu with '設定' (Settings) selected.

- 3.3. GitLab CI/CD 利用ステップ

## 3. GitLab CI/CDの利用

## 3.3. GitLab CI/CD利用ステップ

- 本ガイドでは、以下のGitリポジトリのソースコードを利用し、Open Libertyで稼働するJavaアプリケーションのコンテナビルドしてOpenShiftクラスター上へデプロイする流れを説明する

Deploying microservices to Kubernetes

<https://github.com/OpenLiberty/guide-kubernetes-intro>

- 本ガイドでは、以下のビルドおよびデプロイ処理をGitLab CI/CDによって自動化する
  - Gitリポジトリからのソースコードの取得
  - MavenによるJavaアプリケーションのビルド
  - コンテナイメージのビルドとレジストリーへの格納
  - OpenShiftへのデプロイ
  - デプロイ後のテスト実行

## 3.3. GitLab CI/CD利用ステップ – 利用の流れ

GitLab CI/CD利用の流れは以下の通り

1. リポジトリ作成とソースコードの格納
2. プロジェクト変数の追加
3. .gitlab-ci.yml ファイルの作成
4. Gitリポジトリへのコミット
5. 結果の確認

## 3.3. GitLab CI/CD利用ステップ – 利用の流れ

GitLab CI/CD利用の流れは以下の通り

1. リポジトリ作成とソースコードの格納
2. プロジェクト変数の追加
3. .gitlab-ci.yml ファイルの作成
4. Gitリポジトリへのコミット
5. 結果の確認

### 3.3. GitLab CI/CD利用ステップ-1.リポジトリ作成とソースコードの格納

ビルドおよびデプロイで使用するファイル用のリポジトリ（プロジェクト）を作成し、ソースコードやKubernetesのマニフェスト・ファイル(yaml)を格納する

- 本ガイドでは「2.インストール」にて導入したGitLabsサーバーに、新規リポジトリ「guide-kubernetes-intro」を作成し、以下のリポジトリのファイルをコピーする

Deploying microservices to Kubernetes

<https://github.com/OpenLiberty/guide-kubernetes-intro>

### 3.3. GitLab CI/CD利用ステップ-1.リポジトリ作成とソースコードの格納

1. GitLabに管理者としてログインし、以下のリンク先の手順を参考に空のプロジェクト「guide-kubernetes-intro」を作成する

GitLab日本語マニュアル -プロジェクトの作成

<https://gitlab-docs.creationline.com/ee/gitlab-basics/create-project.html>

2. Macの場合ターミナル、Windowsの場合はPowerShellを開く

```
cd <Gitリポジトリ作成ディレクトリ>
```

3. Git リポジトリを作成するディレクトリへ移動（cd）する

4. 以下のコマンドを実行してコピー元リポジトリをcloneする

```
git clone https://github.com/OpenLiberty/guide-kubernetes-intro.git
```

5. cloneしたリポジトリのディレクトリに移動する

```
cd guide-kubernetes-intro
```

### 3.3. GitLab CI/CD利用ステップ -1.リポジトリ作成とソースコードの格納

6. Gitのユーザー名とメールアドレスを設定する。設定する情報は、Gitでの変更者の記録に使用される

```
git config --global user.name "自分の姓と名の英語表記"  
git config --global user.email "メールアドレス"
```

7.以下のコマンドで既存の接続先Gitリポジトリの登録名をoriginからold-originへ変更する

```
git remote rename origin old-origin
```

8.以下のコマンド1.で新規作成したリポジトリを接続先に追加する

```
git remote add https://<GitLabのURL>/<プロジェクト名またはユーザー名>/guide-kubernetes-intro.git
```

– 実行例

```
$ git remote add origin https://gitlab.mycluster-tok02-b-889642-d85540d5ac7eb4665421ec4a86b8d368-0000.jp-tok.containers.appdomain.cloud/gitlab-qit/guide-kubernetes-intro.git
```

### 3.3. GitLab CI/CD利用ステップ -1.リポジトリ作成とソースコードの格納

9. 以下のコマンドで、新規作成したリポジトリへコードをアップロード（git push）する

```
git push -u origin --all
```

- ログインが求められた場合には、ユーザー名とパスワード（2段階認証の場合はアクセストークン。取得方法は[GitLabマニュアル](#)を参照）を入力する
- git push時に“SSL certificate problem: unable to get local issuer certificate”エラーが発生する場合は、後続の「Tips1. 自己署名証明書を使用するGitLabへのgit操作に失敗する場合の対応」を参照

### 3.3. GitLab CI/CD利用ステップ -1.リポジトリ作成とソースコードの格納

10. ブラウザーでGitLabへログインして作成したリポジトリにソースコードが格納されたことを確認する

The screenshot shows the GitLab web interface for the 'guide-kubernetes-intro' repository. The page includes a navigation sidebar on the left, a top navigation bar, and a main content area. A notification banner at the top indicates that the Auto DevOps pipeline has been enabled. The repository page shows the project name, ID, and statistics (293 commits, 1 branch, 0 tags, 532KB files, 532KB storage). Below this, there are buttons for cloning and a merge pull request card. At the bottom, there is a table listing files and their latest commit details.

名前	最新コミット	最終更新
.github/workflows	Update test.yml and triggerConversion.yml...	3 weeks ago
finish	Merge staging to prod - Version update (#...	2 weeks ago
scripts	Merge staging to prod - Version update (#...	2 weeks ago

### 3.3. GitLab CI/CD利用ステップ-1.リポジトリ作成とソースコードの格納

本ガイドの例では、以下の修正を行った

1. prodブランチからdevelopブランチを作成する
2. developブランチのfinishディレクトリーに補足資料のkubernetes\_develop.yaml を保存する
3. developブランチへ変更をpushする

## 3.3. GitLab CI/CD利用ステップ – 利用の流れ

GitLab CI/CD利用の流れは以下の通り

1. リポジトリ作成とソースコードの格納
2. **プロジェクト変数の追加**
3. .gitlab-ci.yml ファイルの作成
4. Gitリポジトリへのコミット
5. 結果の確認

## 3.3. GitLab CI/CD利用ステップ -2.プロジェクト変数の追加

複数のパイプラインやジョブで利用するデータをプロジェクト変数として登録する。  
また、APIキーなどの秘密情報をマスキングする場合にも変数を利用する

1. ブラウザーから、プロジェクトの管理者権限のあるユーザーでGitLabへログインする
2. 1.で作成したプロジェクトを開く

The screenshot shows the GitLab web interface for a project named 'guide-kubernetes-intro'. The interface includes a navigation sidebar on the left with options like 'プロジェクトの概要', '詳細', 'アクティビティ', 'リリース', 'リポジトリ', 'イシュー', 'マージリクエスト', '要求事項', 'CI/CD', 'セキュリティとコンプライアンス', 'オペレーション', and 'Packages & Registries'. The main content area displays the project name, ID (8), and statistics (293 commits, 1 branch, 0 tags, 532KB files, 532KB storage). A notification at the top states: 'The Auto DevOps pipeline has been enabled and will be used if no alternative CI configuration file is found.' Below this, there are buttons for '設定' and '詳しい情報'. The interface also shows a merge pull request from 'OpenLiberty/staging' and a table of recent commits.

名前	最新コミット	最終更新
.github/workflows	Update test.yml and triggerConversion.yml...	3 weeks ago

## 3.3. GitLab CI/CD利用ステップ -2.プロジェクト変数の追加

3. 左側のメニューから「設定」の上へカーソルを移動し、表示された「CI/CD」をクリックする

The screenshot shows the GitLab interface for a project named 'guide-kubernetes-intro'. The left sidebar contains a menu with '設定' (Settings) highlighted. A dropdown menu is open from '設定', with 'CI/CD' selected and highlighted in red. The main content area displays the project name, project ID (8), and various statistics (293 commits, 1 branch, 0 tags, 532KB files, 532KB storage). Below this, there are buttons for '履歴', 'ファイルを検索', 'Web IDE', and 'クローン'. A merge pull request #178 is visible, authored by Gilbert Kwan 2 weeks ago. At the bottom, a table lists recent commits:

前	最新コミット	最終更新
.github/workflows	Update test.yml and triggerConversion.yml...	3 weeks ago
finish	Merge staging to prod - Version update (#...	2 weeks ago
scripts	Merge staging to prod - Version update (#...	2 weeks ago

## 3.3. GitLab CI/CD利用ステップ -2.プロジェクト変数の追加

### 4. 表示された画面で変数の「展開」をクリックする

The screenshot shows the GitLab CI/CD Settings page for a project named 'guide-kubernetes-intro'. The page is divided into several sections, each with a title and a description, and a '展開' (Expand) button to the right. The '変数' (Variables) section is highlighted with a red box, indicating it is the target of the next step. The '展開' button for this section is also highlighted with a red box.

gitlab-qit > guide-kubernetes-intro > CI/CD Settings

- 一般のパイプライン** (General Pipeline) - 展開 (Expand)
- Auto DevOps** - 展開 (Expand)  
Automate building, testing, and deploying your applications based on your continuous integration and delivery configuration. [How do I get started?](#)
- Protected environments** - 展開 (Expand)  
Only specified users can execute deployments in a protected environment.
- Runner** - 展開 (Expand)  
Runners are processes that pick up and execute CI/CD jobs for GitLab. [How do I configure runners?](#)
- アーティファクト** (Artifacts) - 展開 (Expand)  
A job artifact is an archive of files and directories saved by a job when it finishes.
- 変数** (Variables) - 展開 (Expand)  
Variables store information, like passwords and secret keys, that you can use in job scripts. [Learn more.](#)

## 3.3. GitLab CI/CD利用ステップ -2.プロジェクト変数の追加

### 5. 「変数を追加」をクリックする

The screenshot shows the GitLab web interface for a project named 'guide-kubernetes-intro'. The left sidebar contains navigation links for 'プロジェクトの概要', 'リポジトリ', 'イシュー', 'マージリクエスト', '要求事項', 'CI/CD', 'セキュリティとコンプライア', 'オペレーション', 'Packages & Registries', '分析', 'Wiki', 'スニペット', and 'サイドバーを隠す'. The main content area is titled 'Runners are processes that pick up and execute CI/CD jobs for GitLab. How do I configure runners?' and includes sections for 'アーティファクト' (Artifacts) and '変数' (Variables). The '変数' section explains that variables store information like passwords and secret keys, and lists 'Protected' and 'Masked' types. A yellow warning box states: 'Environment variables are configured by your administrator to be protected by default.' Below this is a table with columns for '種類', 'キー', '値', 'Protected', 'マスク', and 'Environments'. The table is currently empty with the message 'There are no variables yet.' At the bottom of the page, the '変数を追加' (Add variable) button is highlighted with a red box.

Runners are processes that pick up and execute CI/CD jobs for GitLab. [How do I configure runners?](#)

**アーティファクト** 展開

A job artifact is an archive of files and directories saved by a job when it finishes.

**変数** 折りたたむ

Variables store information, like passwords and secret keys, that you can use in job scripts. [Learn more.](#)

Variables can be:

- Protected:** Only exposed to protected branches or tags.
- Masked:** Hidden in job logs. Must match masking requirements. [Learn more.](#)

Environment variables are configured by your administrator to be **protected** by default.

種類	↑ キー	値	Protected	マスク	Environments
There are no variables yet.					

**変数を追加**

グループ変数(継承)

## 3.3. GitLab CI/CD利用ステップ -2.プロジェクト変数の追加

### 6. 変数のキーや値を設定する

– 例、oc login コマンドでの接続先URLを格納する変数

#### 変数の追加 ×

**キー**

**値**

**タイプ** Environment scope

Variable ⇅ 全て(デフォルト) ▼

**Flags**

Protect variable ?  
保護ブランチや保護タグ上で実行されているパイプラインにのみ変数をエクスポートします。

Mask variable ?  
Variable will be masked in job logs. 正規表現の条件を満たす値が必要です。 [詳しい情報](#)

キャンセル 変数の追加



## 3.3. GitLab CI/CD利用ステップ -2.プロジェクト変数の追加

### 7. 入力後「変数の追加」をクリックする

変数の追加 ×

キー

OC\_URL

値

https://c100-e.jp-tok.containers.cloud.ibm.com:32329

タイプ Environment scope

Variable 全て(デフォルト)

Flags

Protect variable ?  
保護ブランチや保護タグ上で実行されているパイプラインにのみ変数をエクスポートします。

Mask variable ?  
Variable will be masked in job logs. 正規表現の条件を満たす値が必要です。 [詳しい情報](#)

キャンセル 変数の追加

## 3.3. GitLab CI/CD利用ステップ -2.プロジェクト変数の追加

### 8. 変数が追加されたことを確認する

The screenshot shows the GitLab CI/CD configuration page for a project. The left sidebar contains navigation options like 'プロジェクトの概要', 'リポジトリ', 'イシュー', 'マージリクエスト', '要求事項', 'CI/CD', 'セキュリティとコンプライア', 'オペレーション', 'Packages & Registries', '分析', 'Wiki', 'スニペット', and 'メンバー'. The main content area is titled 'Runners are processes that pick up and execute CI/CD jobs for GitLab. How do I configure runners?'. Below this, there are sections for 'アーティファクト' (Artifacts) and '変数' (Variables). The '変数' section explains that variables store information like passwords and secret keys. It lists two types: 'Protected' (only exposed to protected branches or tags) and 'Masked' (hidden in job logs). A yellow box states: 'Environment variables are configured by your administrator to be protected by default.' Below this is a table of environment variables:

種類	↑ キー	値	Protected	マスク	Environments
Variable	OC_APIKEY	*****	×	✓	全て(デフォ...
Variable	OC_URL	*****	×	×	全て(デフォ...

At the bottom of the table, there are buttons for '値を表示する' (Show values) and '変数を追加' (Add variable). Below the table, it says 'グループ変数(継承)' (Group variables (inherit)).

## 3.3. GitLab CI/CD利用ステップ – 利用の流れ

GitLab CI/CD利用の流れは以下の通り

1. リポジトリ作成とソースコードの格納
2. プロジェクト変数の追加
3. **.gitlab-ci.yml ファイルの作成**
4. Gitリポジトリへのコミット
5. 結果の確認

## 3.3. GitLab CI/CD利用ステップ -3. .gitlab-ci.yml ファイルの作成

GitLab CI/CDではリポジトリのルートに配置された.gitlab-ci.ymlというファイルに実行内容を定義する

リポジトリへ変更をコミットすると、このファイルに定義した処理が実行される

.gitlab-ci.ymlには以下のような内容を定義する

- 実行対象のジョブの構造や実行順序
- 実行条件

.gitlab-ci.ymlの内容はGitLab Runnerで実行されるため、事前にGitLab Runnerの導入とGitLabへの接続を完了しておく必要がある

(手順は「2.2 OperatorによるGitLab Runnerのインストールと設定」を参照のこと)

## 3.3. GitLab CI/CD利用ステップ –3. .gitlab-ci.yml ファイルの作成

- 本ガイドでは、以下のリンク先のGitリポジトリのコードに対して.gitlab-ci.ymlにCI/CDパイプラインを定義する例を説明する

Deploying microservices to Kubernetes

<https://github.com/OpenLiberty/guide-kubernetes-intro>

- .gitlab-ci.ymlに定義する処理は以下の通り
  - MavenによるJavaアプリケーションのビルド
  - コンテナイメージのビルドとレジストリーへの格納
  - OpenShiftへのデプロイ
  - デプロイ後のテスト実行

## 3.3. GitLab CI/CD利用ステップ -3. .gitlab-ci.yml ファイルの作成

### ■ .gitlab-ci.yml ファイルの作成

.gitlab-ci.ymlはテキストベースのyml形式のファイルであるため、エディターなどで作成し、リポジトリのルートディレクトリにファイル名を「.gitlab-ci.yml」として保存する。

GitLabのテンプレートを利用して作成することも可能であるため、以降手順を説明する

1.パイプライン作成対象のGitリポジトリにWebブラウザでアクセスする

2.画面上部のリストから、ファイルを格納するブランチを選択する

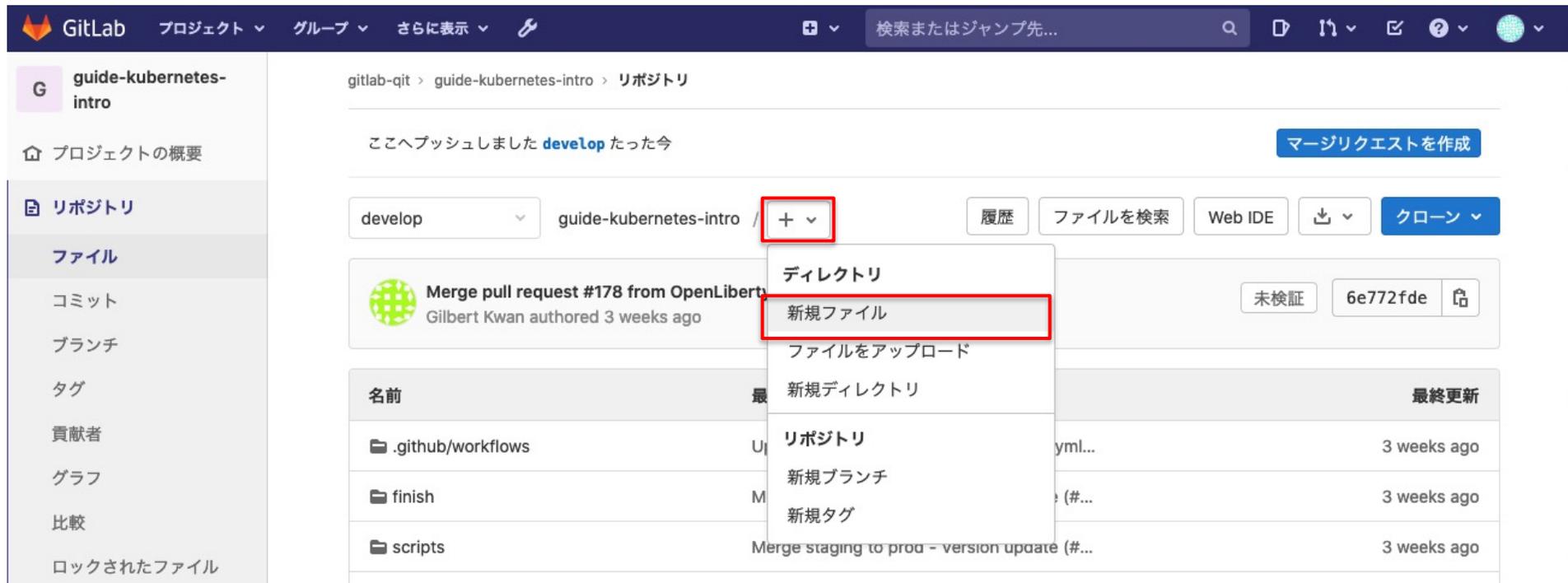
- ブランチの作成が必要な場合には、左側の「リポジトリ」>「ブランチ」で画面を開き「新規ブランチ」をクリックして作成してから選択する



## 3.3. GitLab CI/CD利用ステップ -3. .gitlab-ci.yml ファイルの作成

### ■ .gitlab-ci.yml ファイルの作成

3. ブランチ名およびプロジェクト名の右側の「+」アイコンをクリックし、「新規ファイル」をクリックする



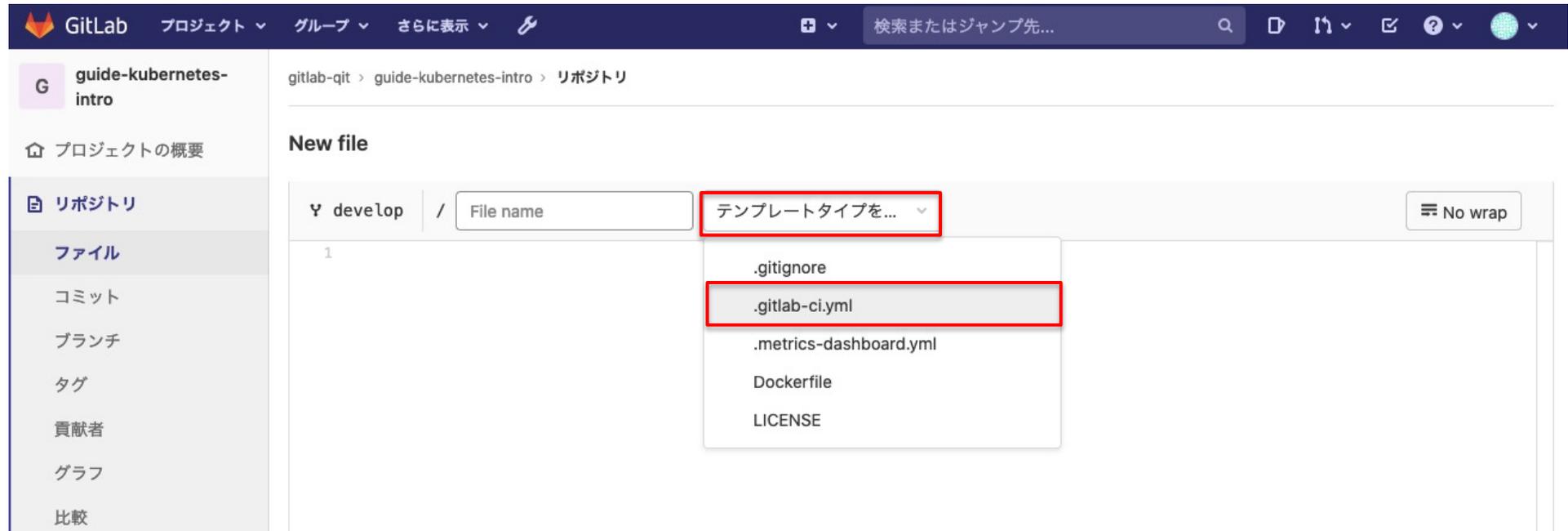
The screenshot shows the GitLab web interface for a project named 'guide-kubernetes-intro'. The current branch is 'develop'. A dropdown menu is open next to the project name, and the option '新規ファイル' (New File) is highlighted. The interface also shows a merge pull request from OpenLiberty and a table of files in the repository.

名前	最終更新
 .github/workflows	3 weeks ago
 finish	3 weeks ago
 scripts	3 weeks ago

## 3.3. GitLab CI/CD利用ステップ -3. .gitlab-ci.yml ファイルの作成

### ■ .gitlab-ci.yml ファイルの作成

#### 4. 「テンプレートタイプを...」 のリストから 「.gitlab-ci.yml」 をクリックする



The screenshot shows the GitLab web interface for creating a new file. The breadcrumb path is 'gitlab-qit > guide-kubernetes-intro > リポジトリ'. The 'New file' dialog is open, showing a 'File name' input field and a 'テンプレートタイプを...' dropdown menu. The dropdown menu is open, displaying a list of template types: '.gitignore', '.gitlab-ci.yml', '.metrics-dashboard.yml', 'Dockerfile', and 'LICENSE'. The '.gitlab-ci.yml' option is highlighted with a red box. The 'develop' branch is selected, and the 'No wrap' option is visible on the right.

## 3.3. GitLab CI/CD利用ステップ -3. .gitlab-ci.yml ファイルの作成

### ■ .gitlab-ci.yml ファイルの作成

#### 5. 「テンプレートを適用」のリストから処理内容に合う項目ををクリックする

The screenshot shows the GitLab web interface for creating a new file. The breadcrumb path is 'gitlab-qt > guide-kubernetes-intro > リポジトリ'. The 'New file' dialog is open, showing the file name as '.gitlab-ci.yml' and the branch as 'develop'. The 'テンプレートを適用' (Apply template) dropdown menu is open, displaying a list of templates. The 'Android-Fastlane' template is currently selected. The list of templates includes: 5-Minute-Production-App, Android, Android-Fastlane, Auto-DevOps, Bash, C++, and Chef. The 'No wrap' button is visible on the right side of the dialog.

## 3.3. GitLab CI/CD利用ステップ -3. .gitlab-ci.yml ファイルの作成

### ■ .gitlab-ci.yml ファイルの作成

6. 選択項目の上部に文字を入力すると、入力した内容でフィルターして表示される

The screenshot shows the GitLab web interface for creating a new file. The breadcrumb path is 'gitlab-qt > guide-kubernetes-intro > リポジトリ'. The 'New file' dialog is open, showing the branch 'develop' and the file name '.gitlab-ci.yml'. A search bar for templates is visible, with 'Maven' entered and highlighted in a red box. Below it, a dropdown menu shows 'Maven' as the selected option. Another example is shown below, with 'OpenShift' entered and highlighted in a red box, and 'OpenShift' shown as the selected option in the dropdown menu. The interface also includes a 'No wrap' button and a search bar at the top.

### 3.3. GitLab CI/CD利用ステップ -3. .gitlab-ci.yml ファイルの作成

#### ■ .gitlab-ci.yml ファイルの作成

7.例として「Bash」を選択する。このテンプレートではメッセージを出力するのみだが、.gitlab-ci.yml の実行の流れを確認できる。

The screenshot shows the GitLab web interface for creating a new file. The breadcrumb path is 'develop / .gitlab-ci.yml'. A dropdown menu for 'テンプレートを適用' (Apply template) is open, showing a list of templates. The 'Bash' template is highlighted with a red box. The main editor area shows the content of the selected template, which includes instructions and a sample build script.

```

1 # This file is a template, and might need editing before it works on
2 # see https://docs.gitlab.com/ee/ci/yaml/README.html for all availab
3
4 # you can delete this line if you're not using Docker
5 image: busybox:latest
6
7 before_script:
8   - echo "Before script section"
9   - echo "For example you might run an update here or install a buil
10  - echo "Or perhaps you might print out some debugging details"
11
12 after_script:
13   - echo "After script section"
14   - echo "For example you might do some cleanup here"
15
16 build1:
17   stage: build
18   script:
19     - echo "Do your build here"
20
21 ..
  
```

## 3.3. GitLab CI/CD利用ステップ -3. .gitlab-ci.yml ファイルの作成

### ■ .gitlab-ci.yml ファイルの作成

#### 8.適宜編集する。

例として、GitLabに自己署名証明書を利用して暫定的にSSL証明書の検査を無効にする場合、変数として以下の設定を追加する

```
variables:  
  GIT_SSL_NO_VERIFY: "true"
```



The screenshot shows the GitLab web interface for creating a new file. The breadcrumb navigation is 'gitlab-qit > guide-kubernetes-intro > リポジトリ'. The 'New file' section shows the file path as 'develop / .gitlab-ci.yml'. The file content is as follows:

```
1 # This file is a template, and might need editing before it works on your project.  
2 # see https://docs.gitlab.com/ee/ci/yaml/README.html for all available options  
3  
4 variables:  
5   GIT_SSL_NO_VERIFY: "true"  
6  
7 # you can delete this line if you're not using Docker  
8 image: busybox:latest  
9
```

## 3.3. GitLab CI/CD利用ステップ -3. .gitlab-ci.yml ファイルの作成

### ■ .gitlab-ci.yml ファイルの作成

9.編集が終わったら、画面下部でコミットメッセージの入力と、ブランチの選択を行い、「Commit changes」をクリックする



```
20 stage: build
21 script:
22   - echo "Do your build here"
23
24 test1:
25   stage: test
26   script:
27     - echo "Do a test here"
28     - echo "For example run a test suite"
29
30 test2:
31   stage: test
32   script:
33     - echo "Do another parallel test here"
34     - echo "For example run a lint test"
35
36 deploy1:
37   stage: deploy
38   script:
39     - echo "Do your deploy here"
40
```

コミットメッセージ: Add new file

ターゲットブランチ: develop

Commit changes

Cancel

## 3.3. GitLab CI/CD利用ステップ -3. .gitlab-ci.yml ファイルの作成

### ■ .gitlab-ci.yml ファイルの記述例

- スクリプトで使用する変数の定義

GIT\_SSL\_NO\_VERIFY: "true"では、検証のため、GitLabサーバーへ接続時のSSL証明書の確認を無効にしている。  
正しい証明書の設定については、[GitLabマニュアル- Providing a custom certificate for accessing GitLab](#) を参照のこと。

```
variables:  
GIT_SSL_NO_VERIFY: "true"  
MAVEN_OPTS: "-Dhttps.protocols=TLSv1.2 -Dmaven.repo.local=${CI_PROJECT_DIR}/.m2/repository -  
Dorg.slf4j.simpleLogger.log.org.apache.maven.cli.transfer.Slf4jMavenTransferListener=WARN -  
Dorg.slf4j.simpleLogger.showDateTime=true -Djava.awt.headless=true"  
MAVEN_CLI_OPTS: "--batch-mode --errors --fail-at-end --show-version -DinstallAtEnd=true -DdeployAtEnd=true"  
SYSTEM_APP_NAME: "system"  
INVENTORY_APP_NAME: "inventory"  
OC_PROJECT_NAME: "fsayaka-guide-kubernetes-intro"
```

MAVEN\_OPTS、MAVEN\_CLI\_OPTSについては、[mavenのパイプラインのサンプル](#)を参考に設定している

## 3.3. GitLab CI/CD利用ステップ –3. .gitlab-ci.yml ファイルの作成

### ■ .gitlab-ci.yml ファイルの記述例

- 実行対象のステージの定義。上から実行する順番に記載する
- 後続のジョブの定義で、ジョブが所属するステージを設定する

```
stages:  
- build  
- package  
- deploy  
- test
```

## 3.3. GitLab CI/CD利用ステップ -3. .gitlab-ci.yml ファイルの作成

### ■ .gitlab-ci.yml ファイルの記述例

- MavenによるJavaアプリケーションのビルドジョブ

```
maven-build:
  image: maven:3.3.9-jdk-8
  stage: build
  script:
    - cd finish
    - 'mvn $MAVEN_CLI_OPTS clean package'
  artifacts:
    paths:
      - finish/system/target
      - finish/inventory/target
    expire_in: 1 week
  cache:
    paths:
      - .m2/repository
```

image:に使用するmavenのバージョンに合わせたコンテナイメージを指定する

stage:に、ジョブが所属するステージを指定する

script:以下にジョブで実行する処理を記載する  
Mavenによるビルドの場合は、ビルドスクリプトで実行するようなmvnコマンドを記載すれば良い

artifacts:以下に、後続のジョブへ共有したいビルド成果物（warなど）のパスを記載する

cache:以下に、アプリのビルドで使用するプラグインなどの依存ファイルのパスを指定する

## 3.3. GitLab CI/CD利用ステップ -3. .gitlab-ci.yml ファイルの作成

### ■ .gitlab-ci.yml ファイルの記述例

- コンテナイメージのビルドとレジストリーへの格納

- サンプルでは「system」と「inventory」の2つのコンテナを作成するため、それぞれ定義する

```

container-build-system:
  stage: package
  image:
    name: gcr.io/kaniko-project/executor:debug
    entrypoint: [""]
  script:
    - mkdir -p /kaniko/.docker
    - echo
    "{¥"auths¥":{¥"$CI_REGISTRY¥":{¥"username¥":¥"$CI_REGISTRY_USER¥",¥"password¥":¥"$CI_REGISTRY_PAS
    SWORD¥"}}}" > /kaniko/.docker/config.json
    - /kaniko/executor --context $CI_PROJECT_DIR/finish/system --dockerfile
    $CI_PROJECT_DIR/finish/system/Dockerfile --destination
    $CI_REGISTRY_IMAGE/system:$CI_COMMIT_REF_SLUG --skip-tls-verify
  
```

image:に使用するコンテナイメージのビルドコマンドを実行できるコンテナイメージを指定する

script:に、コンテナイメージのビルド処理を記述する。詳細は[GitLabマニュアル -Use kaniko to build Docker images](#) を参照のこと

## OpenShift on IBM Cloudのコンテナとして稼働するGitLab Runnerにおけるコンテナ・イメージ・ビルドの考慮点

- コンテナ内でコンテナのビルドコマンドを実行する方法する場合は、端末や仮想サーバー上でビルドをする場合と比較して、利用するビルドツールや権限の選択が難しくなる
- OpenShift on IBM CloudのワーカーノードではDockerデーモンが起動していないため、Docker-in-Docker イメージでのビルドは不可
- Docker-in-Docker以外の選択肢として、下記リンク先にkanikoによる方法が公開されている。本ガイドでは下記ドキュメントを参照し、kanikoを使用してビルドするスクリプトを記載している。  
– [参考 : GitLabマニュアル-Use kaniko to build Docker images](#)
- その他のツールとしてpodmanやbuidahを利用するという選択肢も考えられるが、OpenShiftの場合rootやprivileged権限でのコンテナの起動に制約があるため、十分な調査と検証が必要となる

## 3.3. GitLab CI/CD利用ステップ -3. .gitlab-ci.yml ファイルの作成

### ■ .gitlab-ci.yml ファイルの記述例

- OpenShiftへのデプロイ

```
.openshift_job:
  before_script:
    #- export PATH=$CI_PROJECT_DIR:$PATH
    - export HOME=/tmp
    #- oc login $OC_URL --token=$OC_TOKEN --insecure-skip-tls-verify
    - oc login $OC_URL -u apikey -p $OC_APIKEY --insecure-skip-tls-verify
    - oc project $OC_PROJECT_NAME
```

oc loginやoc projectは前提の処理として別のジョブに before\_scriptとして記載（このように分けることは必須ではない）

```
openshift-deploy:
  image: openshift/origin-cli
  extends: .openshift_job
  stage: deploy
  script:
    - oc apply -f $CI_PROJECT_DIR/finish/kubernetes_develop.yaml
    - oc rollout status deployment/inventory-deployment --timeout=240s
    - oc rollout status deployment/system-deployment --timeout=240s
```

image:にOpenShiftのCLIであるocコマンドを実行できるイメージを指定

script:にoc applyやoc rollout statusなどロールアウトの処理を記載する

## 3.3. GitLab CI/CD利用ステップ -3. .gitlab-ci.yml ファイルの作成

### ■ .gitlab-ci.yml ファイルの記述例

- デプロイ後のテスト実行

```
integration-test:  
  image: maven:3.3.9-jdk-8  
  stage: test  
  script:  
    - cd start  
    - 'mvn $MAVEN_CLI_OPTS failsafe:integration-test -Dsystem.context.root=/dev'
```

mvnコマンドでデプロイ後のテストを実装している場合は、  
stage:testで実行可能

## 3.3. GitLab CI/CD利用ステップ – 利用の流れ

GitLab CI/CD利用の流れは以下の通り

1. リポジトリ作成とソースコードの格納
2. プロジェクト変数の追加
3. .gitlab-ci.yml ファイルの作成
4. Gitリポジトリへのコミット
5. 結果の確認

## 3.3. GitLab CI/CD利用ステップ – 4. Gitリポジトリへのコミット

Gitリポジトリのルートにgitlab-ci.ymlファイルを格納すると、以降そのリポジトリにコミットがpushされるたびにgitlab-ci.ymlに定義した処理が実行される

実行対象のブランチを限定する方法については以下のリンク先を参照のこと

– 参考：GitLab CI/CD パイプライン設定リファレンス - only/except (基本)

<https://gitlab-docs.creationline.com/ee/ci/yaml/#onlyexcept-basic>

## 3.3. GitLab CI/CD利用ステップ – 利用の流れ

GitLab CI/CD利用の流れは以下の通り

1. リポジトリ作成とソースコードの格納
2. プロジェクト変数の追加
3. .gitlab-ci.yml ファイルの作成
4. Gitリポジトリへのコミット
5. **結果の確認**

## 3.3. GitLab CI/CD利用ステップ – 5. 結果の確認

GitLab CI/CDの実行結果はGitLabのWeb UIから確認する

1. ブラウザーでgitlab-ci.ymlを格納したGitリポジトリへアクセスする
2. 左側のメニューから「CI/CD」をクリックし「パイプライン」をクリックする

The screenshot displays the GitLab web interface for a repository named 'gitlab-git > guide-kubernetes-intro > Repository'. The left sidebar contains a navigation menu with 'CI/CD' highlighted in red, and a sub-menu 'パイプライン' (Pipeline) also highlighted in red. The main content area shows a commit by Sayaka Fukushima with a message 'This GitLab CI configuration is valid.' and a preview of the .gitlab-ci.yml file content. The file content includes a 'before\_script' section with a command to echo a message.

```
before_script:
  - echo "For example you might run an update here or install a build dependency"
```

## 3.3. GitLab CI/CD利用ステップ – 5. 結果の確認

### 3. パイプラインの一覧が表示される

コミット列に実行対象のブランチ名やコミットの情報が表示される

The screenshot displays the GitLab CI/CD pipeline overview page. The page shows a list of pipelines with columns for status, pipeline ID, trigger, commit, and stages. A pipeline with ID #124 is highlighted in red, showing it is in progress (実行中) and triggered by a commit on the develop branch (develop -> 1b7c048b) with the commit message 'Update .gitlab-ci.yml'. The interface includes a search bar, filters, and a sidebar with navigation options like 'プロジェクトの概要', 'リポジトリ', 'イシュー', 'マージリクエスト', '要求事項', 'CI/CD', 'パイプライン', 'Editor', 'ジョブ', 'スケジュール', 'Test Cases', and 'セキュリティとコンプライア'.

### 3.3. GitLab CI/CD利用ステップ – 5. 結果の確認

4.パイプラインの実行内容を確認するにはパイプライン列の番号をクリックする

The screenshot shows the GitLab CI/CD interface for a project named 'guide-kubernetes-intro'. The main content area displays a list of pipelines. The pipeline with ID #125 is currently running and is highlighted with a red box. The pipeline #124 is shown as failed.

ステータス	パイプライン	トリガー	コミット	ステージ	アクション
実行中	#125 最新		develop -> f8fc2e96 Update .gitlab-ci.yml		In progress
失敗	#124		develop -> 1b7c048b Update .gitlab-ci.yml		00:00:23 2 minutes ago

## 3.3. GitLab CI/CD利用ステップ – 5. 結果の確認

5.パイプラインの詳細として実行対象のジョブなどが表示される

The screenshot displays the GitLab CI/CD interface for a pipeline. The left sidebar shows the project structure, with 'CI/CD' and 'パイプライン' (Pipeline) selected. The main content area shows the pipeline details for 'Update .gitlab-ci.yml'.

At the top, the pipeline is identified as 'Pipeline #125' triggered 57 seconds ago by 'Sayaka Fukushima'. A red '実行のキャンセル' (Cancel execution) button and a '削除' (Delete) button are visible.

The pipeline title is 'Update .gitlab-ci.yml'. Below it, the commit information is shown: '4 job develop の場合 (queued for 12 second)', 'latest' branch, and commit hash 'f8fc2e96'. A message states 'No related merge requests found.'

The pipeline status is shown as 'パイプライン Needs ジョブ 4 テスト 0'. The pipeline graph shows three stages: 'Build', 'Test', and 'Deploy'. The 'Build' stage has a job 'build1' with a green checkmark. The 'Test' stage has two jobs, 'test1' and 'test2', both with blue circular icons. The 'Deploy' stage has a job 'deploy1' with a blue circular icon.

## 3.3. GitLab CI/CD利用ステップ – 5. 結果の確認

### 6. ジョブの詳細を確認するには、ジョブ名をクリックする

The screenshot displays the GitLab CI/CD interface for a pipeline named "Update .gitlab-ci.yml". The pipeline is currently in the "実行中" (Running) state, triggered 57 seconds ago by Sayaka Fukushima. The pipeline consists of four jobs: "build1", "test1", "test2", and "deploy1". The "build1" job is highlighted with a red box, indicating it is the current focus. The "test1" and "test2" jobs are in a pending state, and the "deploy1" job is also in a pending state. The interface includes a sidebar with navigation options such as "プロジェクトの概要", "リポジトリ", "イシュー", "マージリクエスト", "要求事項", "CI/CD", "パイプライン", "Editor", "ジョブ", "スケジュール", "Test Cases", "セキュリティとコンプライア", "オペレーション", and "サイドバーを隠す". The main content area shows the pipeline details, including the commit hash "f8fc2e96" and the message "No related merge requests found."

## 3.3. GitLab CI/CD利用ステップ – 5. 結果の確認

### 7. ジョブの詳細情報として、実行時間やログが表示される

The screenshot displays the GitLab CI/CD interface for a project named 'guide-kubernetes-intro'. The left sidebar shows navigation options like 'プロジェクトの概要', 'リポジトリ', 'イシュー', 'マージリクエスト', '要求事項', 'CI/CD', 'パイプライン', 'エディタ', 'ジョブ', 'スケジュール', 'テストケース', 'セキュリティとコンプライアンス', 'オペレーション', and 'サイドバーを隠す'. The main area shows the details for a job named 'build1'. The job status is '成功' (Success) and it took 26 seconds to complete. The logs show the following steps:

```

19 ContainersNotReady: "containers with unready status: [build helper]"
20 Running on runner-jkskbksm-project-8-concurrent-06z2bk via gitlab-runner-runner-8554c7c895-z4gvw...
22 Getting source from Git repository 00:03
23 Fetching changes with git depth set to 50...
24 Initialized empty Git repository in /builds/gitlab-qi/guide-kubernetes-intro/.git/
25 Created fresh repository.
26 Checking out f8fc2e96 as develop...
27 Skipping Git submodules setup
29 Executing "step_script" stage of the job script 00:01
30 $ echo "Before script section"
31 Before script section
32 $ echo "For example you might run an update here or install a build dependency"
33 For example you might run an update here or install a build dependency
34 $ echo "Or perhaps you might print out some debugging details"
35 Or perhaps you might print out some debugging details
36 $ echo "Do your build here"
37 Do your build here
39 Running after_script 00:04
40 Running after script...
41 $ echo "After script section"
42 After script section
43 $ echo "For example you might do some cleanup here"
44 For example you might do some cleanup here
46 Cleaning up file based variables 00:01
48 Job succeeded
  
```

On the right side, the job details are shown, including the duration (26 秒), timeout (1h (project から)), runner (gitlab-runner-runner-8554c7c895-z4gvw (#848)), and the commit (f8fc2e96). The job is associated with pipeline #125 on the develop branch. A dropdown menu shows the job name 'build' and a button '再試行' (Retry) is visible.

## 3.3. GitLab CI/CD利用ステップ – 5. 結果の確認

8. ジョブの実行に失敗している場合は、ログを確認し、gitlab-ci.ymlファイルの修正など対応を行う

The screenshot shows the GitLab CI/CD interface for a project named 'guide-kubernetes-intro'. The left sidebar contains navigation options like 'プロジェクトの概要', 'リポジトリ', 'イシュー', 'マージリクエスト', '要求事項', 'CI/CD', 'パイプライン', 'Editor', 'ジョブ', 'スケジュール', 'Test Cases', 'セキュリティとコンプライア', and 'オペレーション'. The main area displays the execution log for a job named 'build1'. The log shows a series of 'ContainersNotReady' messages, followed by a fatal error: 'fatal: unable to access 'https://gitlab.mycluster-tok02-b-889642-d85540d5ac7eb4665421ec4a86b8d368-0000.jp-tok.containers.appdomain.cloud/gitlab-git/guide-kubernetes-intro.git/': SSL certificate problem: unable to get local issuer certificate'. The job status is 'ERROR: Job failed: command terminated with exit code 1'. The right sidebar shows job details for 'build1', including 'Duration: 23 秒', 'Timeout: 1h (project から)', 'Runner: gitlab-runner-runner-8554c7c895-z4gvw (#848)', and a '再試行' (Retry) button. Below the details, there is a '新規イシュー' (New Issue) button and a section for 'ジョブのアーティファクト' (Job Artifacts) with a commit hash '1b7c048b' and a dropdown menu for 'build'.

```

11 Waiting for pod gitlab-system/runner-jkskbksm-project-8-concurrent-0297w7 to be running, status is Pending
12     ContainersNotReady: "containers with unready status: [build helper]"
13     ContainersNotReady: "containers with unready status: [build helper]"
14 Waiting for pod gitlab-system/runner-jkskbksm-project-8-concurrent-0297w7 to be running, status is Pending
15     ContainersNotReady: "containers with unready status: [build helper]"
16     ContainersNotReady: "containers with unready status: [build helper]"
17 Waiting for pod gitlab-system/runner-jkskbksm-project-8-concurrent-0297w7 to be running, status is Pending
18     ContainersNotReady: "containers with unready status: [build helper]"
19     ContainersNotReady: "containers with unready status: [build helper]"
20 Waiting for pod gitlab-system/runner-jkskbksm-project-8-concurrent-0297w7 to be running, status is Pending
21     ContainersNotReady: "containers with unready status: [build helper]"
22     ContainersNotReady: "containers with unready status: [build helper]"
23 Running on runner-jkskbksm-project-8-concurrent-0297w7 via gitlab-runner-runner-8554c7c895-z4gvw...
24
25 Getting source from Git repository 00:02
26 Fetching changes with git depth set to 50...
27 Initialized empty Git repository in /builds/gitlab-git/guide-kubernetes-intro/.git/
28 Created fresh repository.
29 fatal: unable to access 'https://gitlab.mycluster-tok02-b-889642-d85540d5ac7eb4665421ec4a86b8d368-0000.jp-tok.containers.appdomain.cloud/gitlab-git/guide-kubernetes-intro.git/': SSL certificate problem: unable to get local issuer certificate
30
31 Cleaning up file based variables 00:01
32
33 ERROR: Job failed: command terminated with exit code 1

```

# 4. Tips

## Tips1. 自己署名証明書を使用するGitLabへのgit操作に失敗する場合の対応(1/2)

- SSL自己署名証明書を使用するGitLabに対して、https接続をしてgit操作（clone, pull, pushなど）を行う際に以下のようなエラーが発生する場合、Gitクライアント側への自己署名証明書の登録が必要

– 実行例

```
$ git push -u origin --all
fatal: unable to access 'https://gitlab.mycluster-tok02-b-889642-d85540d5ac7eb4665421ec4a86b8d368-0000.jp-
tok.containers.appdomain.cloud/gitlab-qit/guide-kubernetes-intro.git/': SSL certificate problem: unable to get local issuer certificate
```

### ■ 対応手順

1. GitLabサーバーをインストールしたOpenShiftクラスターにログインし、以下のコマンドでSSL自己署名証明書をダウンロードする（参考：[\(Optional\) Add root CA](#)）

```
oc get secret gitlab-wildcard-tls-ca -ojsonpath='{.data.cfssl_ca}' -n gitlab-system | base64 --decode > GitLabサーバーのホスト名.pem
```

– 実行例

```
oc get secret gitlab-wildcard-tls-ca -ojsonpath='{.data.cfssl_ca}' -n gitlab-system | base64 --decode > gitlab.mycluster-tok02-b-889642-
d85540d5ac7eb4665421ec4a86b8d368-0000.jp-tok.containers.appdomain.cloud.pem
```

## Tips1. 自己署名証明書を使用するGitLabへのgit操作に失敗する場合の対応(2/2)

2.以下のコマンドでgit configの設定に追加する（参考: [Gitマニュアル-http.sslCAInfo](#)）

```
git config --global http.https://Gitサーバーのホスト名.sslcainfo <1. で.pem ファイルを保存したパス>
```

– 実行例

```
$ git config --global http.https://gitlab.mycluster-tok02-b-889642-d85540d5ac7eb4665421ec4a86b8d368-0000.jp-tok.containers.appdomain.cloud.sslcainfo /Users/aha01107/work/GitlabQIT/gitlab.mycluster-tok02-b-889642-d85540d5ac7eb4665421ec4a86b8d368-0000.jp-tok.containers.appdomain.cloud.pem
```

3. 操作対象のGitリポジトリのパスに移動（cd）して、git config -l コマンドを実行し、設定が反映されていることを確認する

– 実行例

```
$ git config -l  
<省略>  
http.https://gitlab.mycluster-tok02-b-889642-d85540d5ac7eb4665421ec4a86b8d368-0000.jp-tok.containers.appdomain.cloud.sslcainfo=/Users/aha01107/work/GitlabQIT/gitlab.mycluster-tok02-b-889642-d85540d5ac7eb4665421ec4a86b8d368-0000.jp-tok.containers.appdomain.cloud.pem  
<省略>
```

## Tips2. 自己署名証明書を使用するGitLabのコンテナ・レジストリー操作のための設定

- 自己署名証明書を使用するGitLabのコンテナ・レジストリーの操作（login, pull, pushなど）を行う際には、クライアント側に証明書の設定が必要になる
  
- OpenShiftクラスターにGitLab OperatorでインストールしたGitLabであれば、レジストリーの証明書は、gitlab-systemプロジェクトのroute/gitlab-registry-<ランダムな値> のCertificateの値を利用すれば良い
  
- クライアント側への証明書の設定手順については以下の情報を参考に実施する
  - docker
    - [dockerドキュメント -証明書をリポジトリのクライアント認証に使用](#)
    - /etc/docker/certs.d ディレクトリの下に、レジストリのホスト名と同じ名前のディレクトリを作成し、.crd拡張子のファイル（例、ca.crt）を作成し、コピーした証明書の値を格納する
  
  - OpenShiftクラスター
    - [ビルドの信頼される認証局の追加設定](#)
    - 端末で /etc/docker/certs.d ディレクトリの下に、レジストリのホスト名と同じ名前のディレクトリを作成し、.crd拡張子のファイル（例、ca.crt）を作成し、コピーした証明書の値を格納した上で、上記リンク先の手順を実施する

## Tips3. GitLabに関する問題の修正依頼方法(1/2)

- GitLabの各種機能のソースコードはGitLab.comのリポジトリに公開されている
- GitLab.comのアカウント登録を行い、対応する機能のリポジトリのissuesページからissue(イシュー、課題)を登録すると、DeveloperやMaintainerから回答が得られる
  - リポジトリによりissueの登録可否は異なると考えられる

- 参考 : GitLab Operatorのissuesページ

<https://gitlab.com/gitlab-org/cloud-native/gitlab-operator/-/issues>

The screenshot shows the GitLab web interface for the 'GitLab Operator' repository. The page title is 'Issues' and it shows 67 open issues. A callout box with a red border and a green arrow points to the 'New issue' button, which is also highlighted with a red box. The callout text reads: '[New issue]をクリックして、issueを登録する。記載内容は次ページを参照のこと'. Below the callout, the list of issues is visible, including titles like 'Define a pattern for validating resource existence' and 'Code linting stage in pipeline is relatively slow'.

## Tips3. GitLabに関する問題の修正依頼方法(2/2)

- issueはGitLabのMarkdown形式での記載が可能
  - [参考 : GitLab Flavored Markdown](#)
  
- 問題の報告の場合の記載例
  - Title (タイトル) に問題が分かりやすい名前をつける
  - Description (記載内容) に以下の内容を含める。それぞれのタイトルは太字にすると見やすい。
    - Summary 何が問題なのかを簡潔に書く
    - Steps to reproduce 問題の再現手順
    - What is the current *bug* behavior? 現在のバグの挙動
    - What is the expected *correct* behavior? 期待する振る舞い
    - GitLab Environment Info GitLabのバージョン、その他コンポーネント (RunnerやOperatorなど) のバージョンなど
  
- 他に登録されているissueを参考に、読みやすい記載を心がける
  - 例 : [本ガイド作成時に発見し、登録したissue](#)
  
- Labelsなどその他の情報はDeveloperにより設定されると考えられるため、登録時には指定しない

## Tips4. 稼働時の消費リソース例

- 無負荷時の kubectl top pod -n gitlab-system

```
% kubectl top pod -n gitlab-system
NAME                                CPU(cores)  MEMORY(bytes)
gitlab-controller-manager-64c76b794c-rh4z8  9m          110Mi
gitlab-gitaly-0                       12m         206Mi
gitlab-gitlab-exporter-79b5757597-b6pcz    3m          22Mi
gitlab-gitlab-shell-6b88fd658d-mbkznz     4m          2Mi
gitlab-minio-0                         0m          9Mi
gitlab-postgresql-0                   8m          60Mi
gitlab-redis-master-0                 29m         21Mi
gitlab-registry-65cb6dc6f5-jgp82        0m          9Mi
gitlab-registry-65cb6dc6f5-jxn72        0m          11Mi
gitlab-runner-controller-manager-5d6bb74795-8hlbx  3m          23Mi
gitlab-runner-runner-8554c7c895-z4gvw    5m          21Mi
gitlab-sidekiq-all-in-1-v1-6685f45688-nr6v6  46m         1154Mi
gitlab-task-runner-95f78cbb6-pf95p      0m          0Mi
gitlab-webservice-default-84dc779c77-k256z  5m          1926Mi
gitlab-webservice-default-84dc779c77-tvtnf  5m          2019Mi
```

# Tips4. 稼働時の消費リソース例

The screenshot displays the Red Hat OpenShift Container Platform console interface. The main content area is titled "Overview" and "Cluster".

**Details:**

- Cluster API Address: <https://c100-8.jp-tok.containers.cloud.ibm.com:32329>
- Cluster ID: d4a88c6d-5ba0-40fa-ae94-386900f8a73
- Provider: IBMCloud
- OpenShift Version: 4.6.34
- Update Channel: Not available

**Status:** Cluster is green (OK), Operators are red (1 degraded). An alert is shown: "Alerts are not configured to be sent to a notification system, meaning that you may not be notified in a timely fashion when important failures occur. Check the OpenShift documentation to learn how to configure notifications with Alertmanager." A "Configure" link is provided.

**Cluster Inventory:**

- 3 Nodes
- 185 Pods
- 20 Storage Classes
- 5 PVCs

**Cluster Utilization (1 Hour):**

Resource	Usage	13:45	14:00	14:15	14:30
CPU	4.8 of 12	6	4	2	
Memory	20.19 GiB of 46.47 GiB	30 GiB	20 GiB	10 GiB	
Filesystem	22.31 GiB of 70.49 GiB	30 GiB	20 GiB	10 GiB	
Network Transfer	6.51 MBps in, 6.6 MBps out	20 MBps	10 MBps		
Pod count	186	200	100		

**Activity:** Recent events include "Stopping container registry-server", "Error: ImagePullBackOff", "Started container registry-server", "Created container registry-server", "Successfully pulled image 'regis...", "Add eth0 [172.30.64.178/32]", "Pulling image 'registry.redhat.io/...", "Successfully assigned openshift...", "Chart value 'cainjectorim...", "Stopping container registry-ser...", "Started container registry-server", "Created container registry-server", "Successfully pulled image 'regis...", "Add eth0 [172.30.64.178/32]", "Pulling image 'registry.redhat.io/...", "Successfully assigned openshift...", "Stopping container registry-ser...", "constraints not satisfiable...", and "Started container registry-server".

# Tips4. 稼働時の消費リソース例

The screenshot displays the Red Hat OpenShift console interface for the 'gitlab-system' project. The left sidebar shows navigation options like Administrator, Home, Overview, Projects, Search, Explore, Events, Operators, Workloads, Networking, Storage, Builds, Monitoring, Compute, User Management, and Administration. The main content area is titled 'Project Details' for 'gitlab-system' and is currently on the 'Overview' tab. It shows the project is 'Active' and provides a summary of resource usage over a 1-hour period.

**Project Details:**

- Name: gitlab-system
- Requester: No requester
- Labels: control-plane=controller-manager, olm.operatorgroup.uid/96f3b69-1d6-4f7f-aL...
- Description: No description

**Inventory:**

- 9 Deployments
- 0 Deployment Configs
- 4 Stateful Sets
- 32 Pods
- 4 PVCs
- 17 Services
- 4 Routes
- 20 Config Maps
- 41 Secrets

**Utilization (1 Hour):**

Resource	Usage	13:45	14:00	14:15	14:30
CPU	142.5m	200m	150m	100m	50m
Memory	5.49 GiB	8 GiB	6 GiB	4 GiB	2 GiB
Filesystem	147 GiB	2 GiB	1 GiB		
Network Transfer	335.6 KiBps in 329.5 KiBps out	400 KiBps	200 KiBps		
Pod count	32	40	20		

**Resource Quotas:**

No resource quotas

**Activity:**

Ongoing: There are no ongoing activities.

Recent Events:

- 14:38 Error: ImagePullBackOff
- 14:33 Back-off pulling image "scan.com..."
- 13:42 Readiness probe failed: comm...
- 12:43 Pulling image "scan.connect.redha..."
- 12:29 Readiness probe failed: comm...

**ありがとうございました**