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 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
    - コードーレビュー
    - 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 Gitaly 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、セキュリティー上の課題がある。
    - 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 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

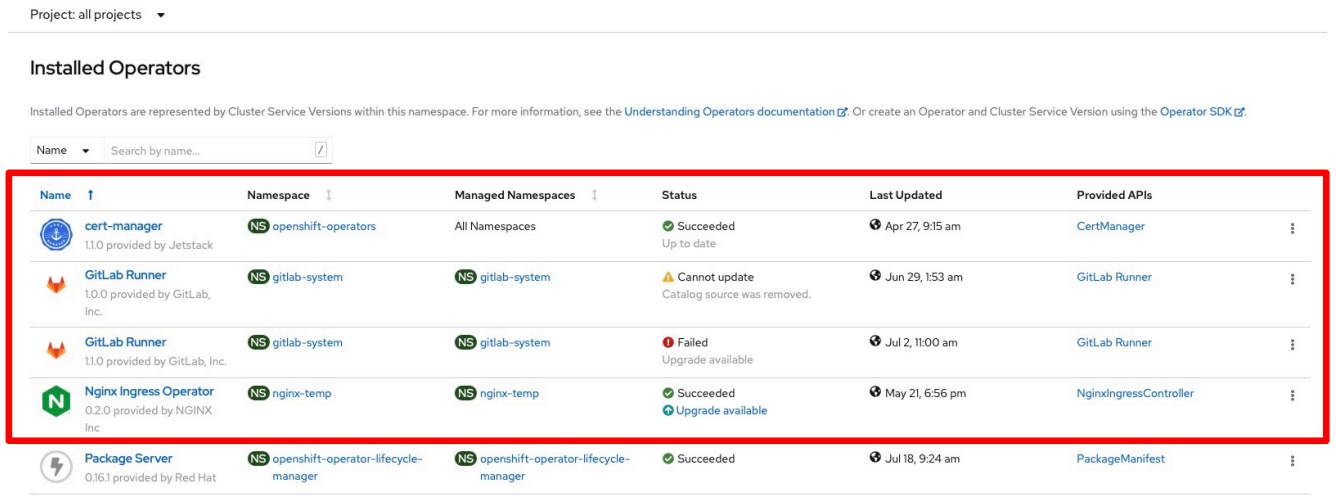
1. OperatorによるGitLabのインストール
2. OperatorによるGitLab Runnerのインストール
3. ライセンスの設定

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のリポジトリーの手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



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

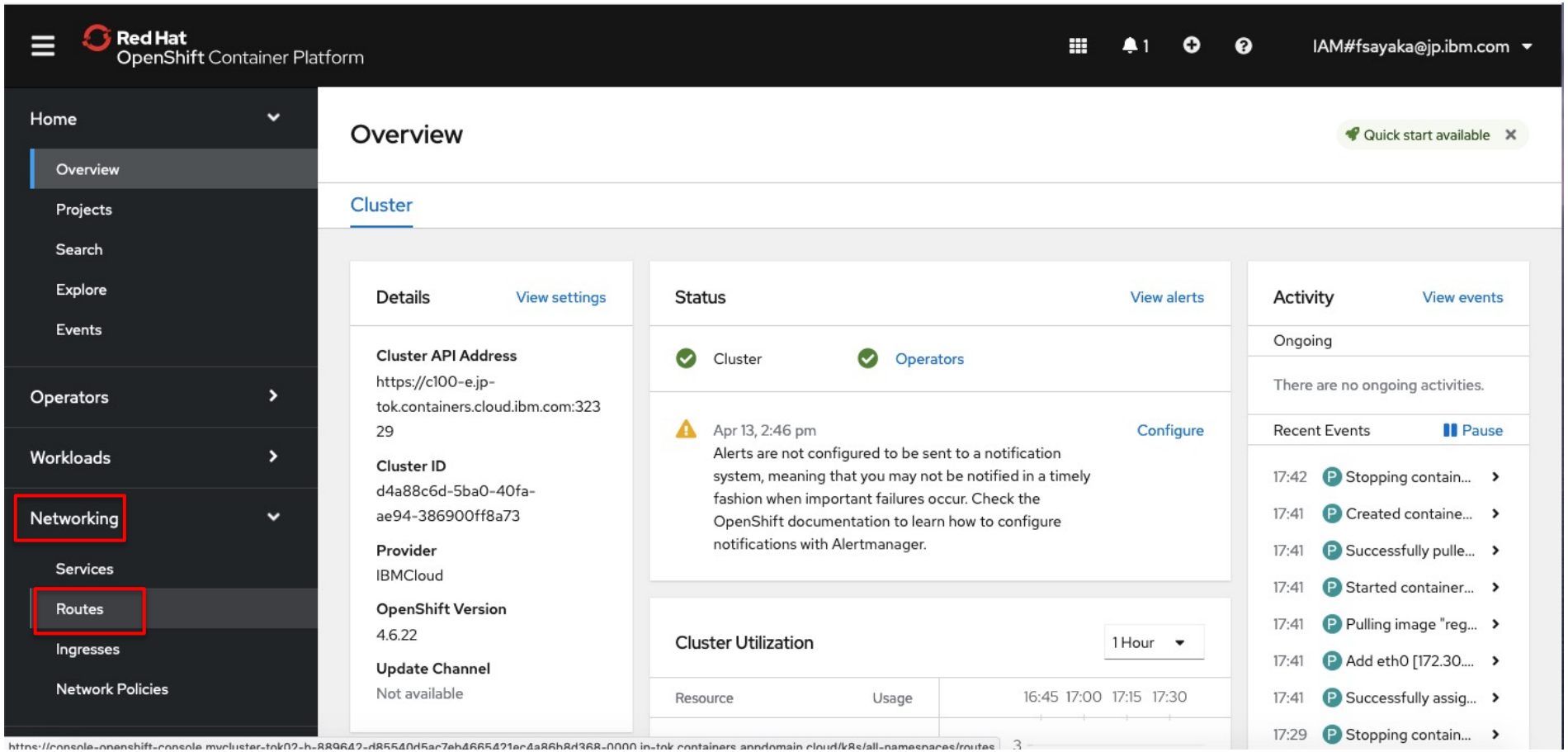
2.1. OperatorによるGitLabのインストール – 導入の流れ

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

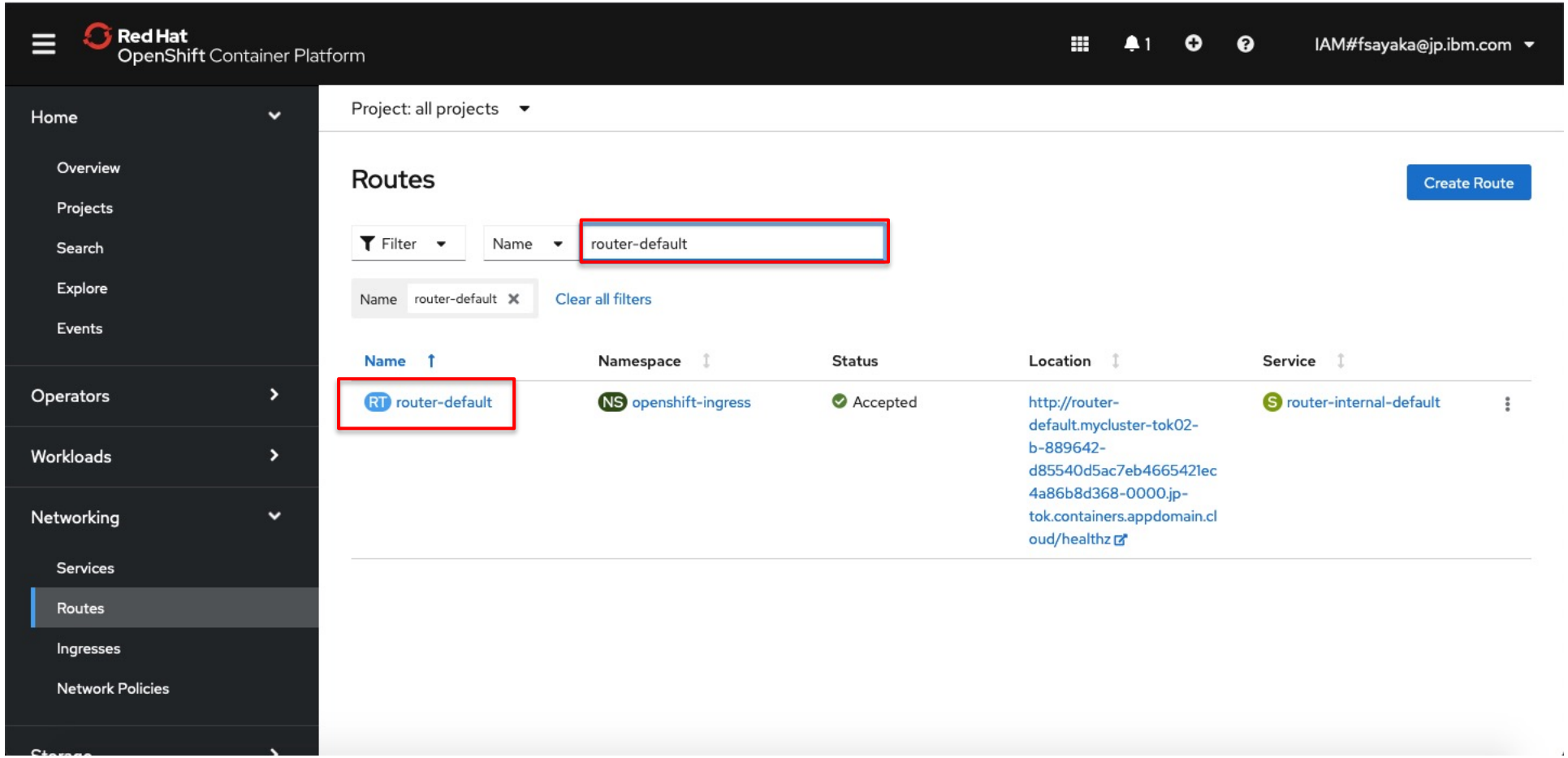
2.1. OperatorによるGitLabのインストール – 2.ドメイン名の準備（default routeの確認）

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



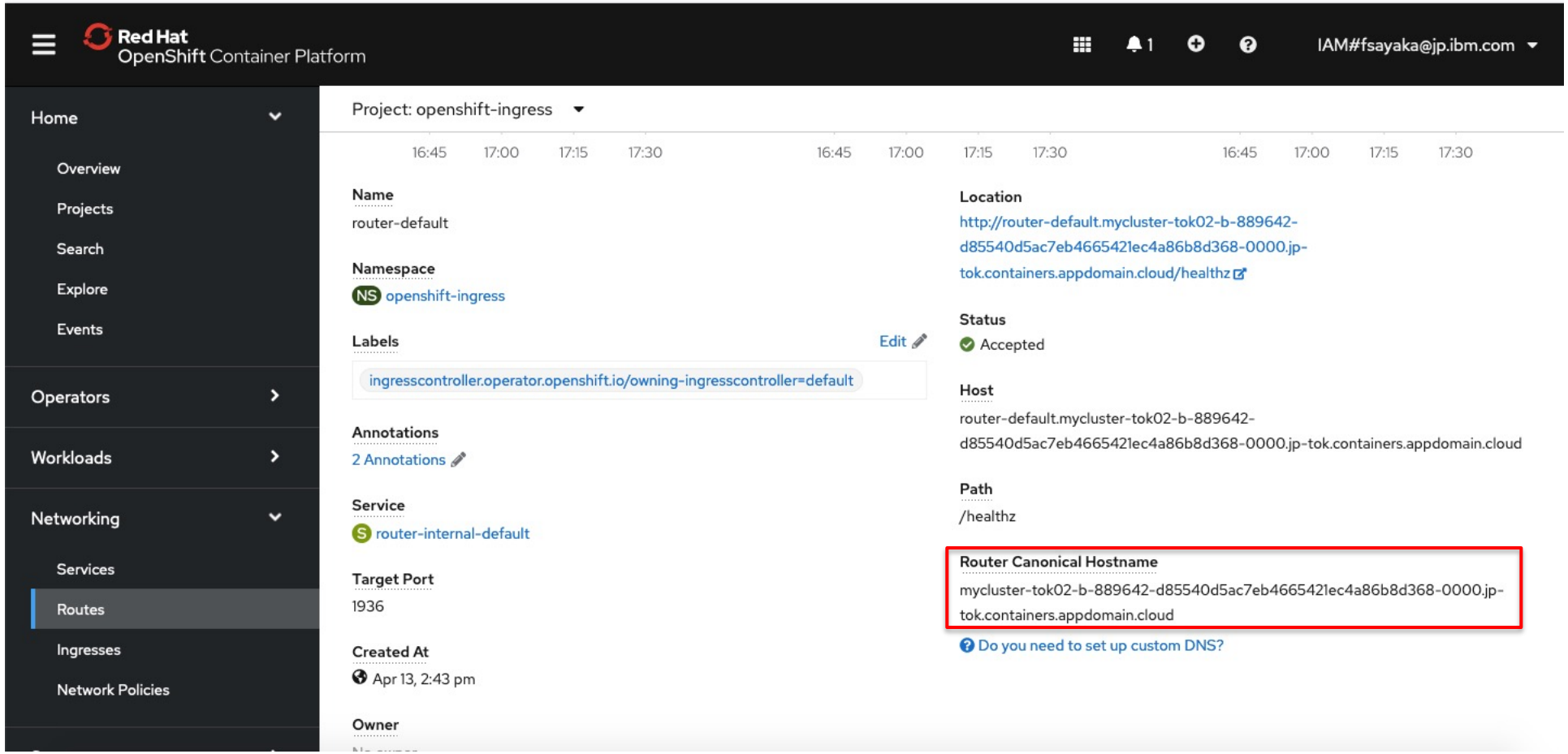
2.1. OperatorによるGitLabのインストール – 2.ドメイン名の準備（default routeの確認）

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



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）

3. 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



2.1. OperatorによるGitLabのインストール – 導入の流れ

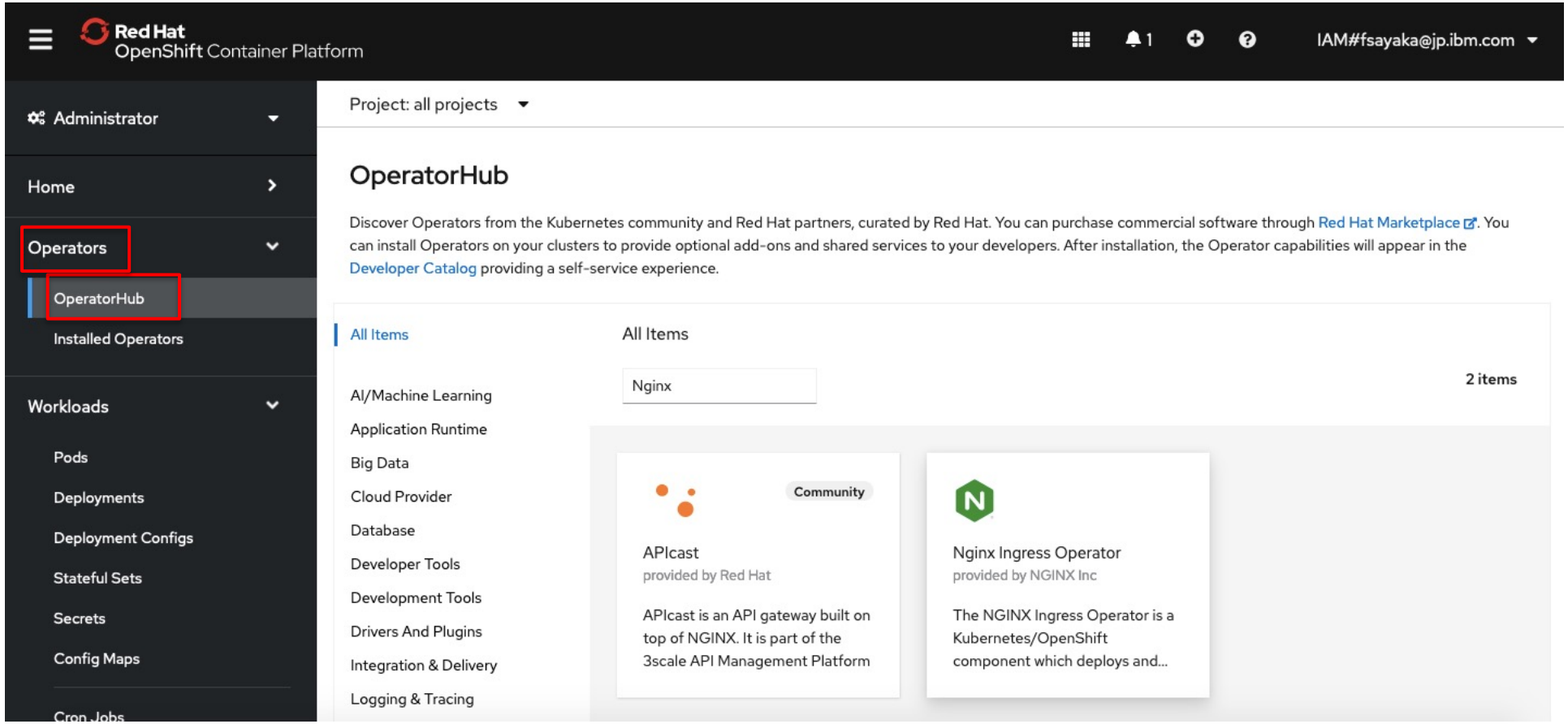
導入の流れは以下の通り

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

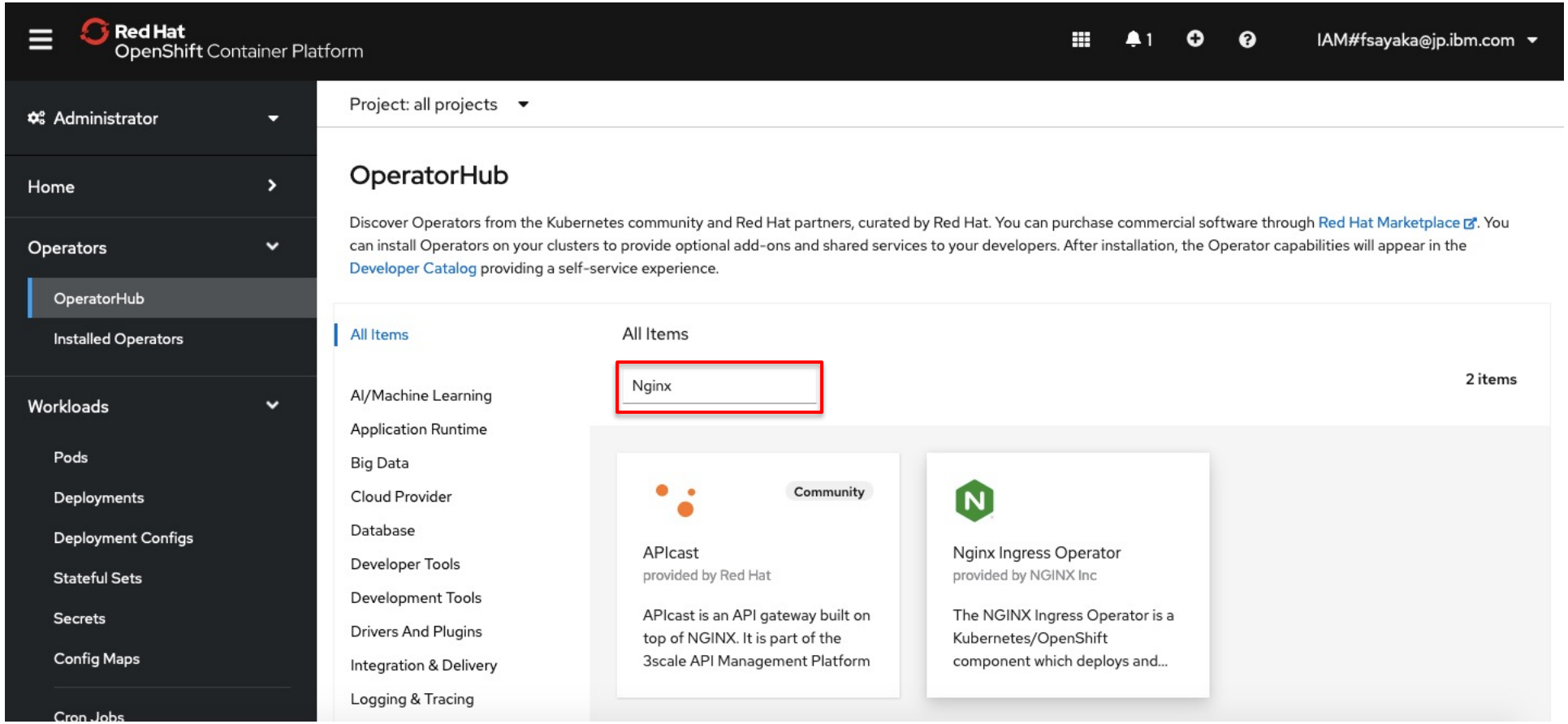
■ 前提要件に記載されているNGINX IngressをOperatorを使用して導入する

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



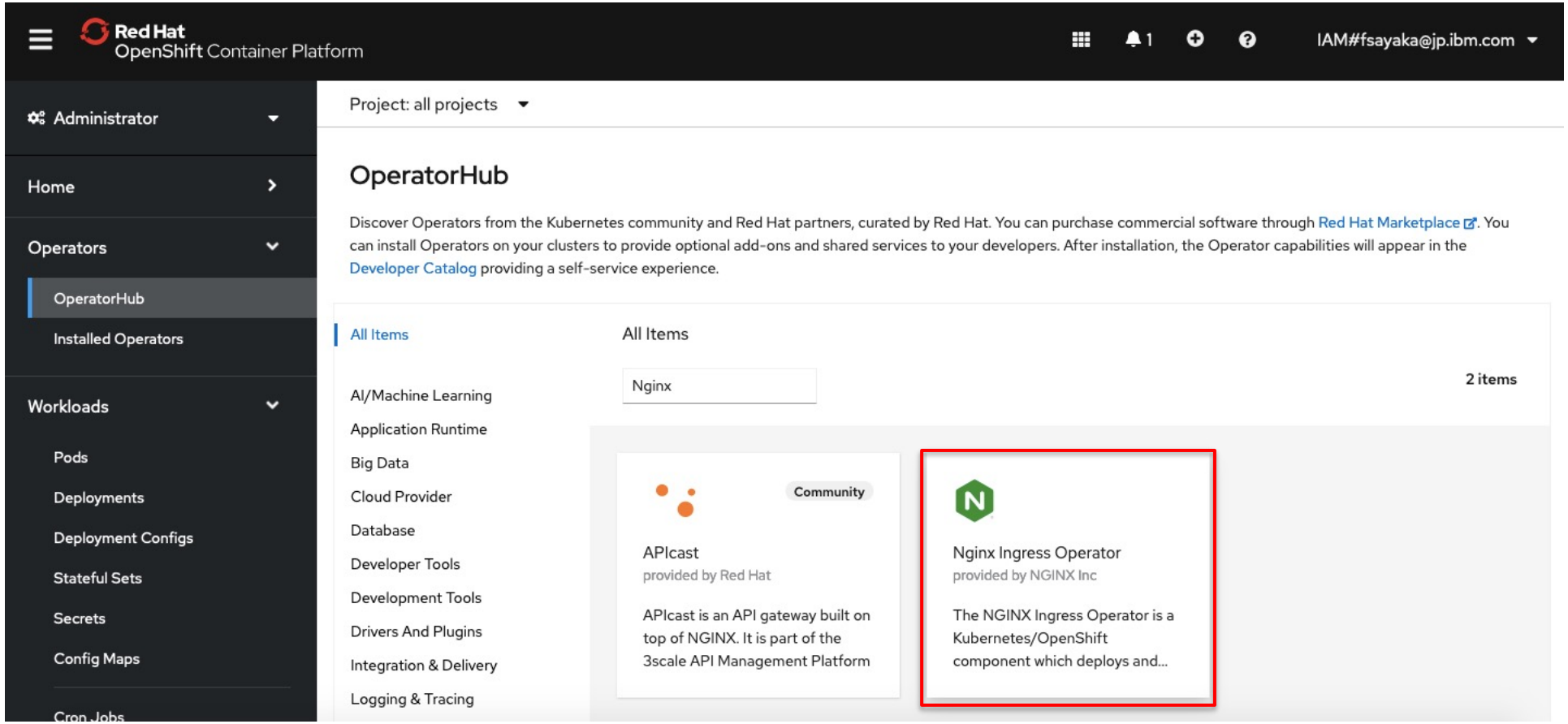
2.1. OperatorによるGitLabのインストール –3.NGINX Ingressの導入

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



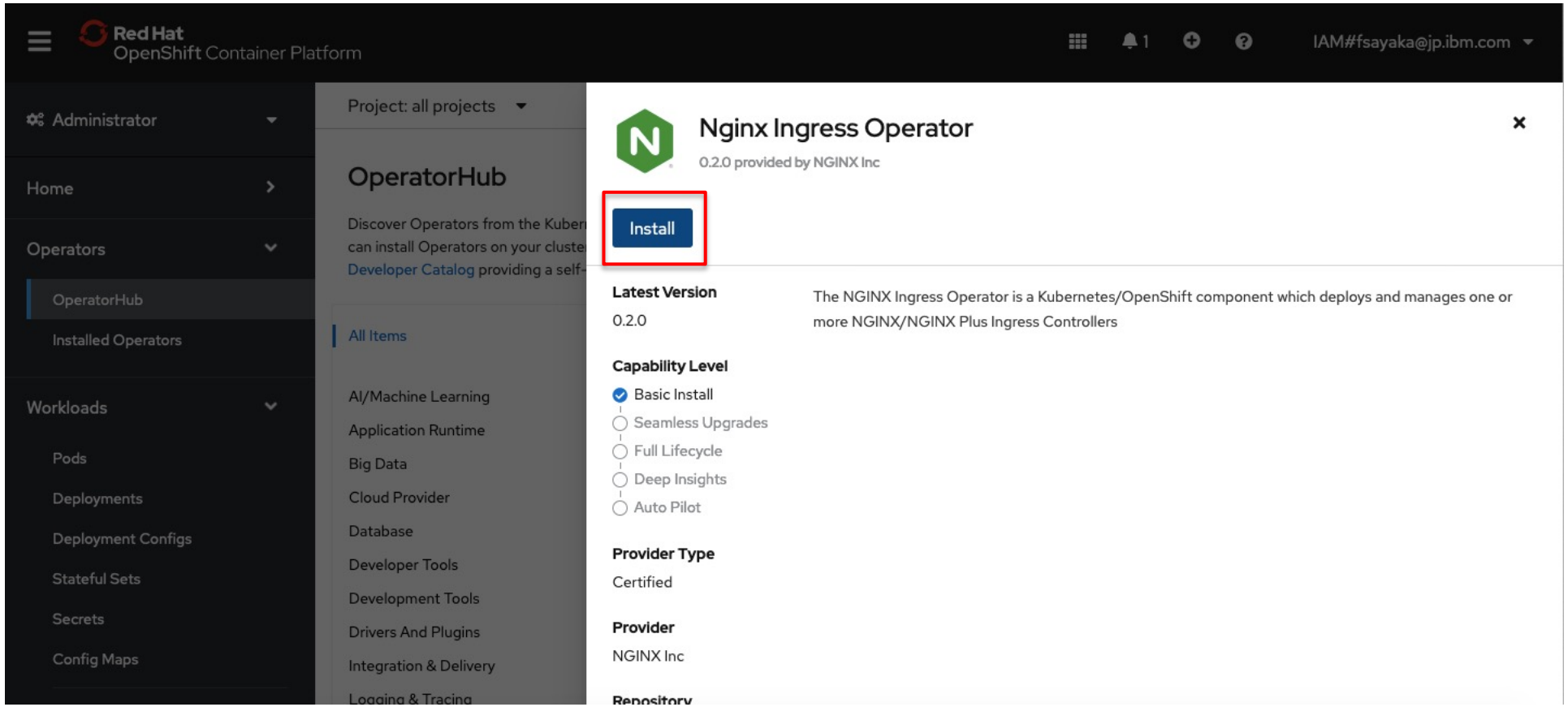
2.1. OperatorによるGitLabのインストール –3.NGINX Ingressの導入

3. Nginx Ingress Operatorをクリックする



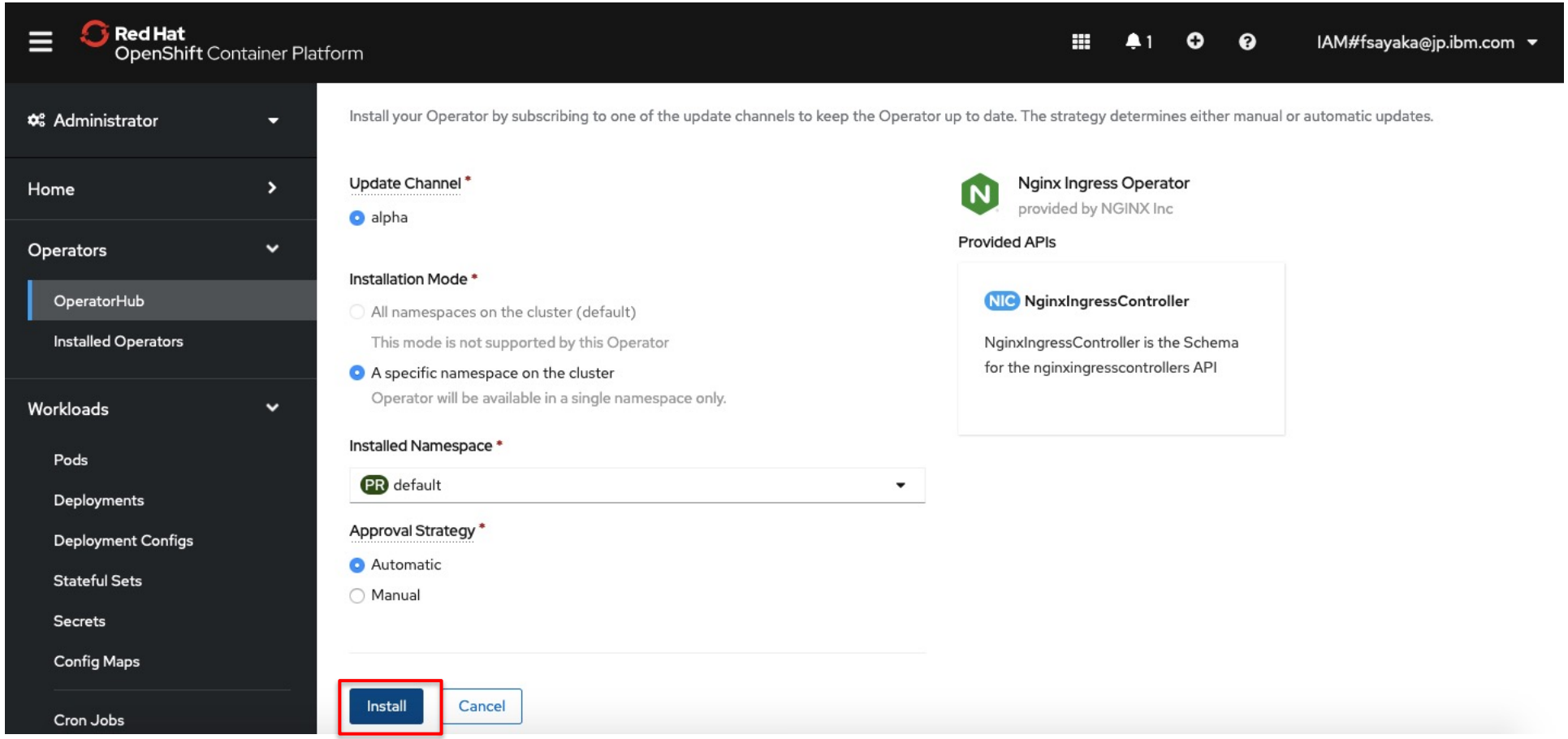
2.1. OperatorによるGitLabのインストール –3.NGINX Ingressの導入

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



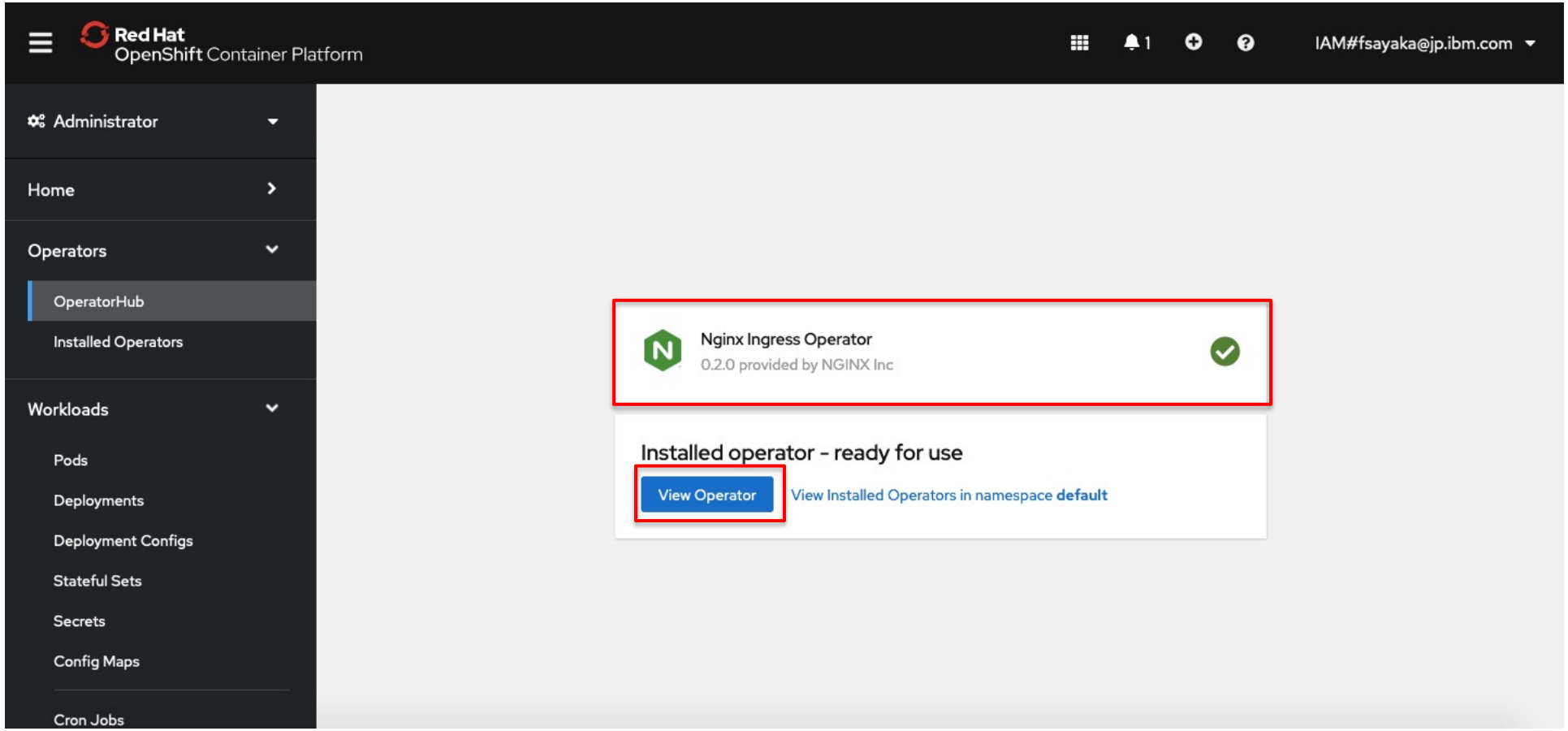
2.1. OperatorによるGitLabのインストール –3.NGINX Ingressの導入

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



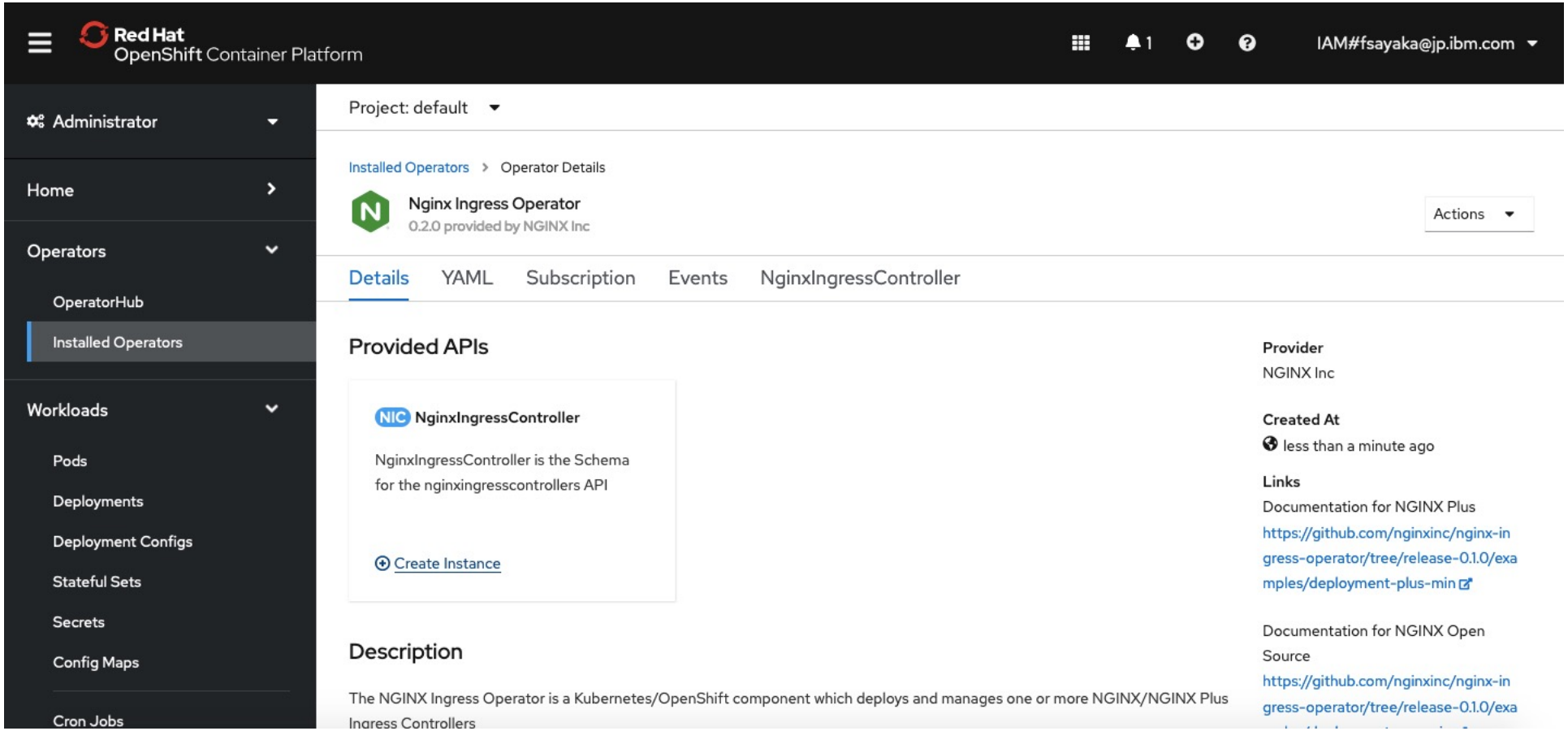
2.1. OperatorによるGitLabのインストール –3.NGINX Ingressの導入

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



- 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

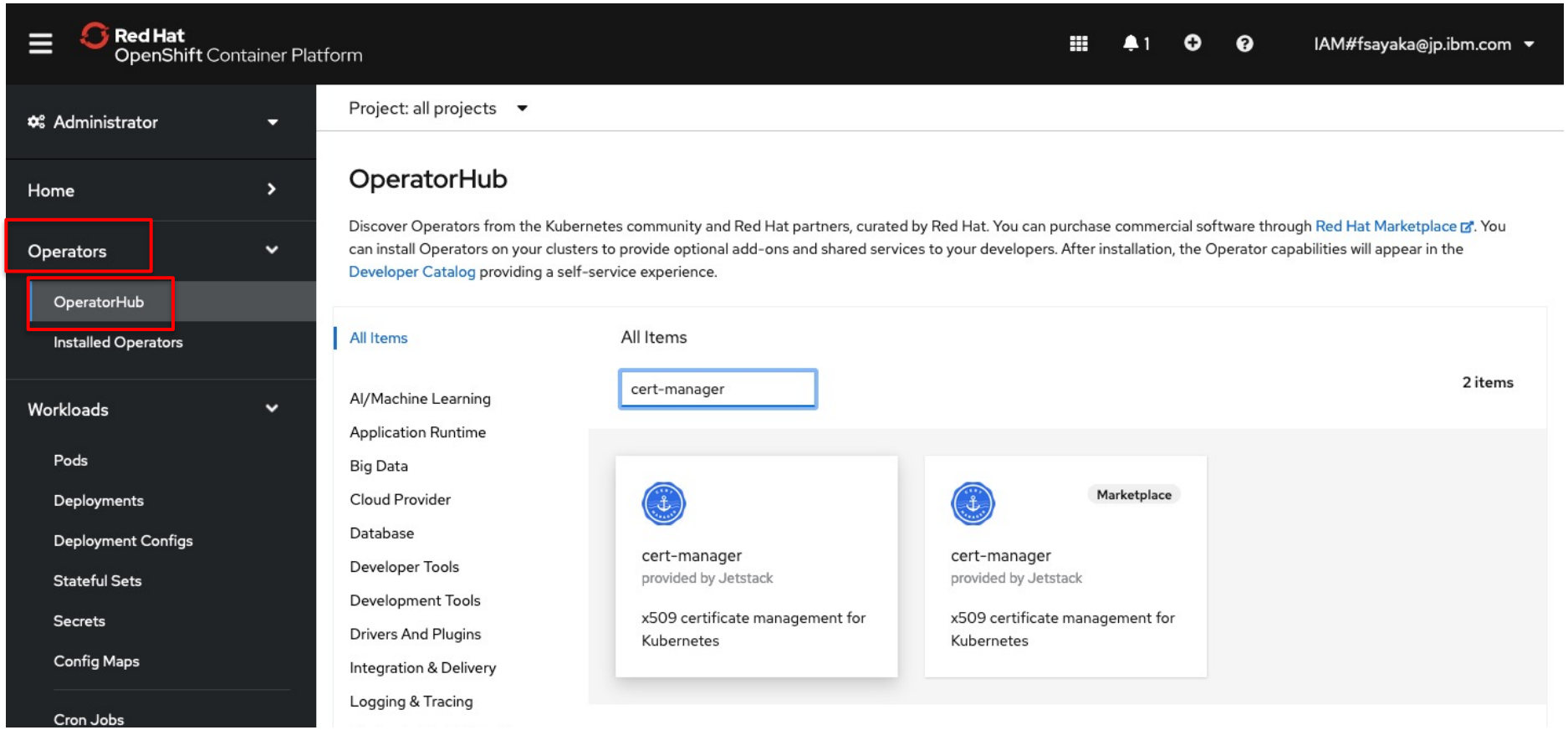
導入の流れは以下の通り

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

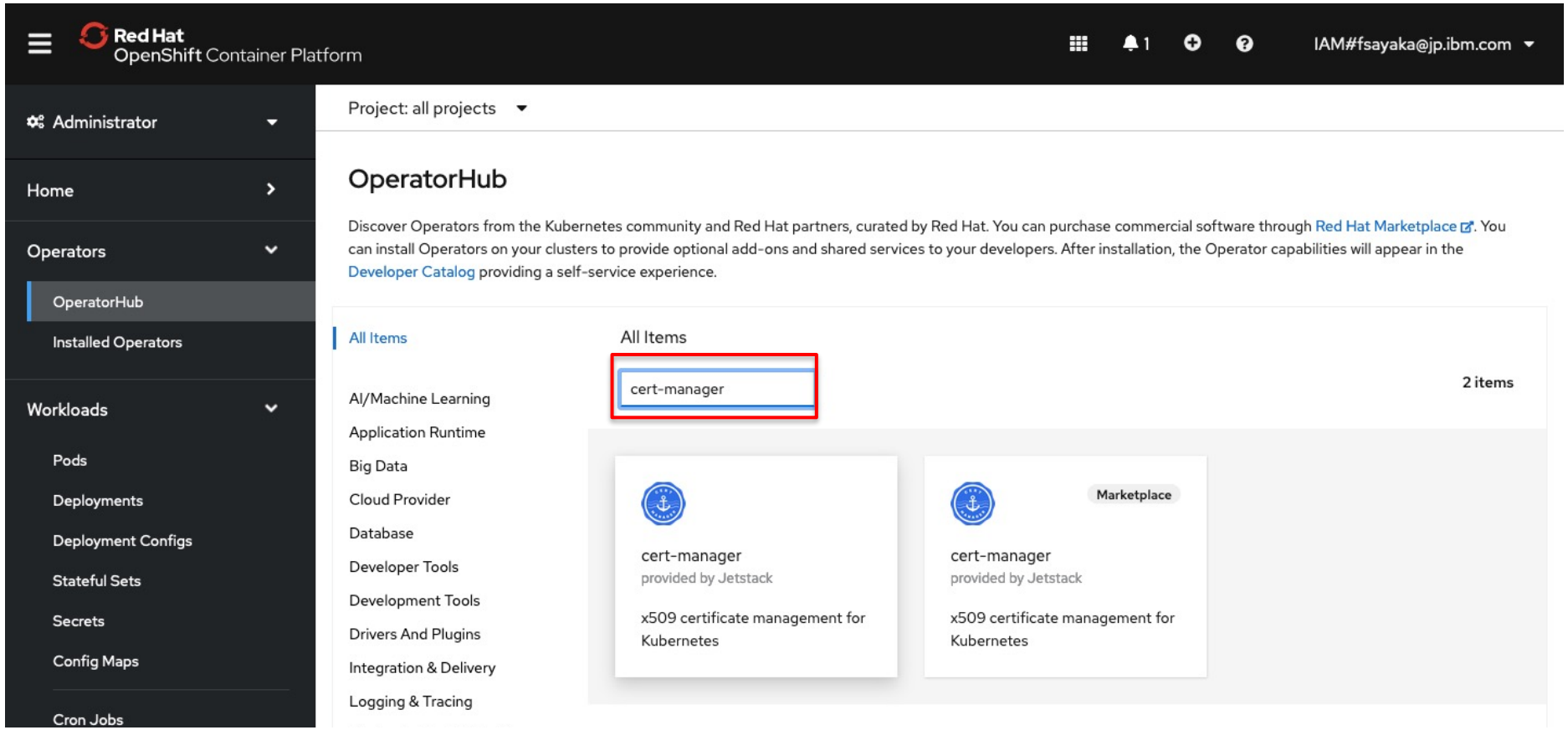
- GitLabやレジストリーへのアクセス時に使用する証明書を作成するためのCert Manager operatorを導入する

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



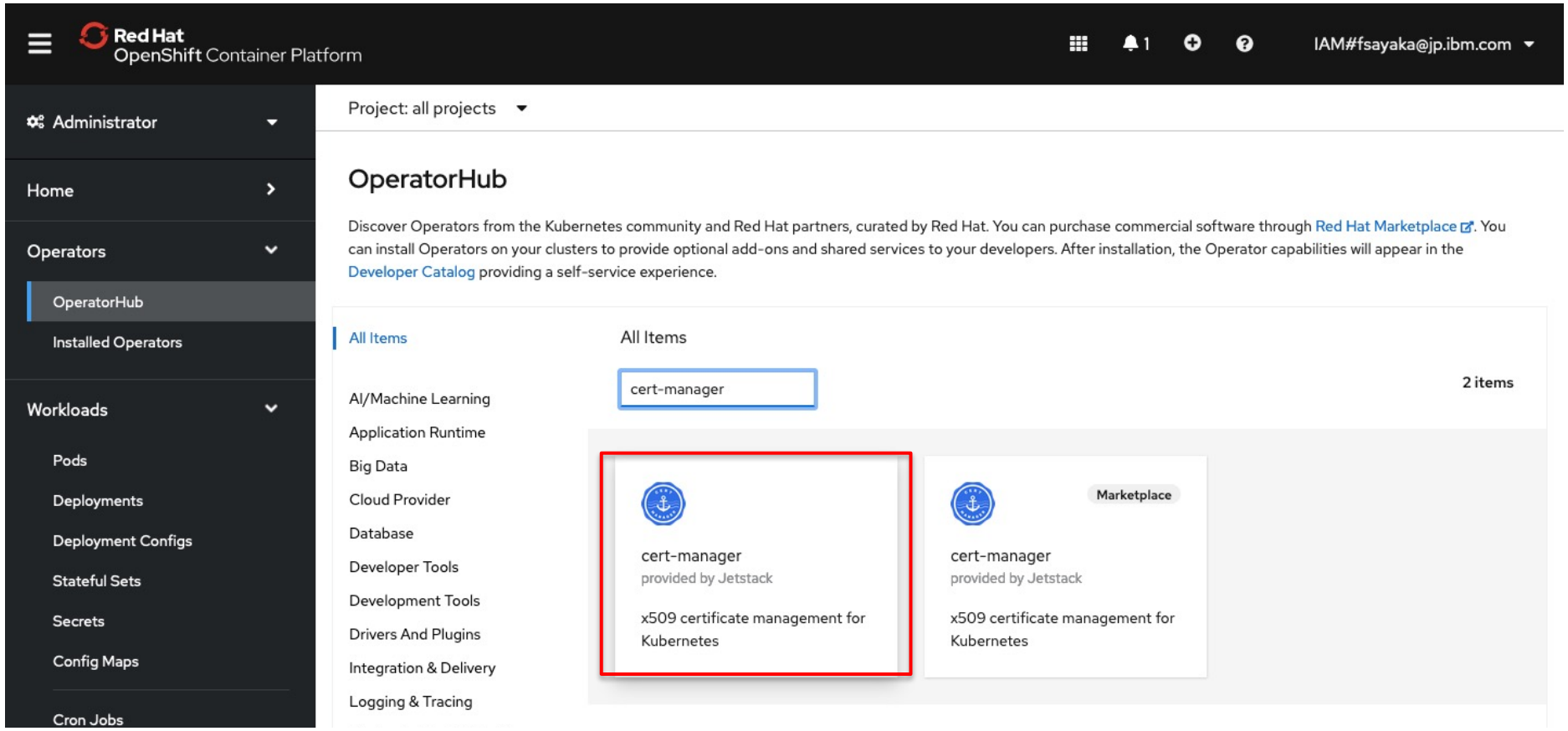
2.1. OperatorによるGitLabのインストール – 4.Cert Manager の導入

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



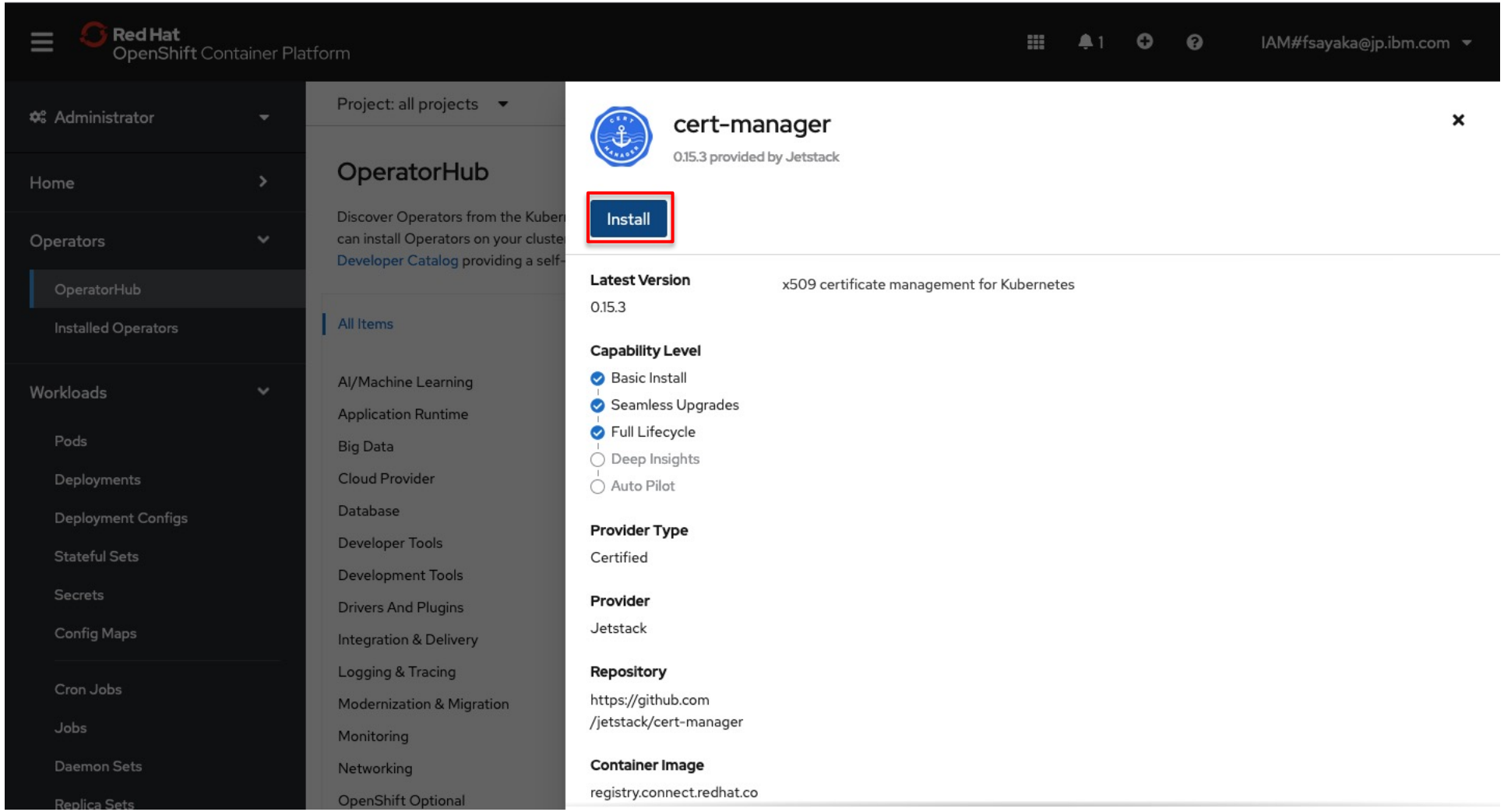
2.1. OperatorによるGitLabのインストール – 4.Cert Manager の導入

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



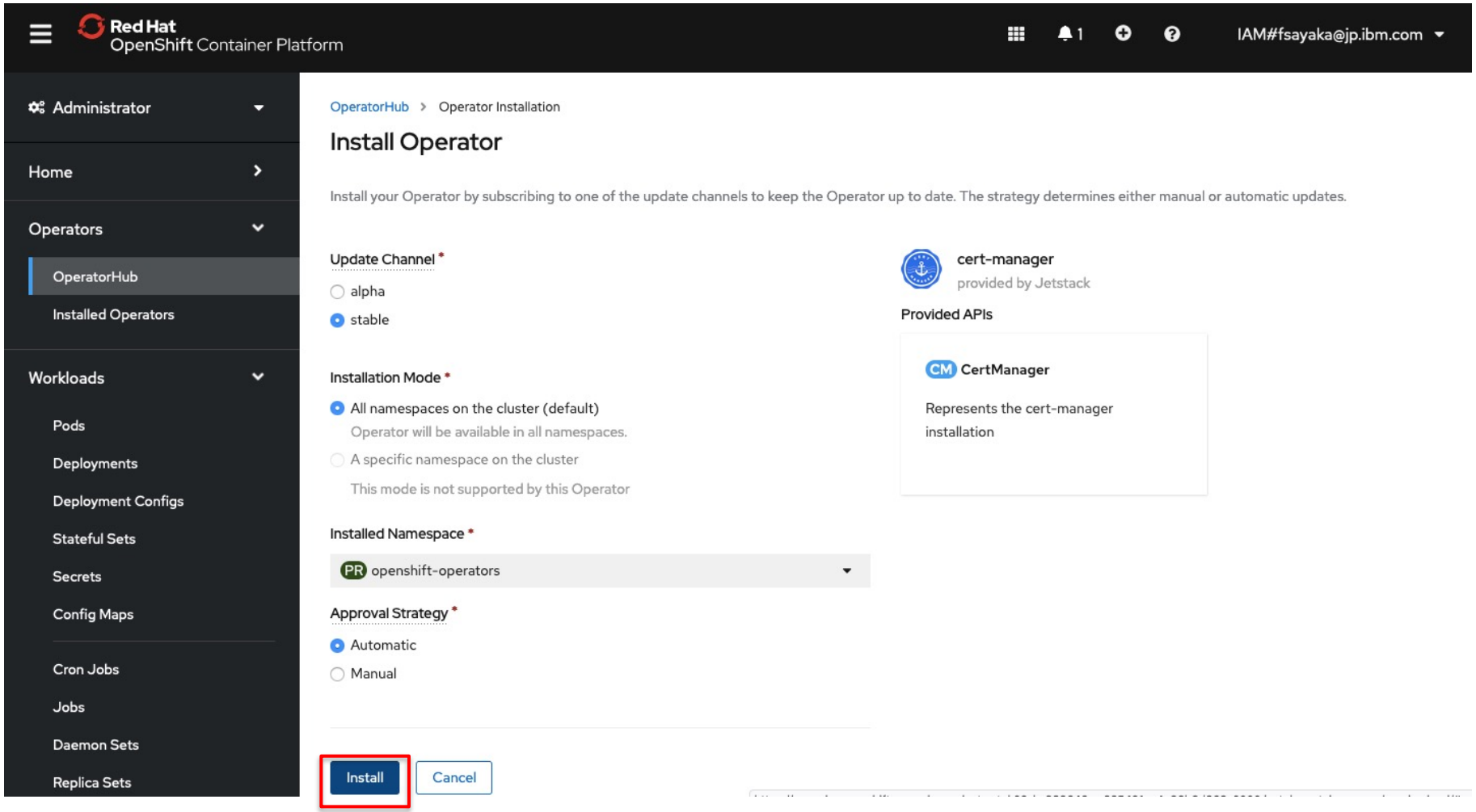
2.1. OperatorによるGitLabのインストール – 4.Cert Manager の導入

4.「cert-manager」の画面で「Install」をクリックする



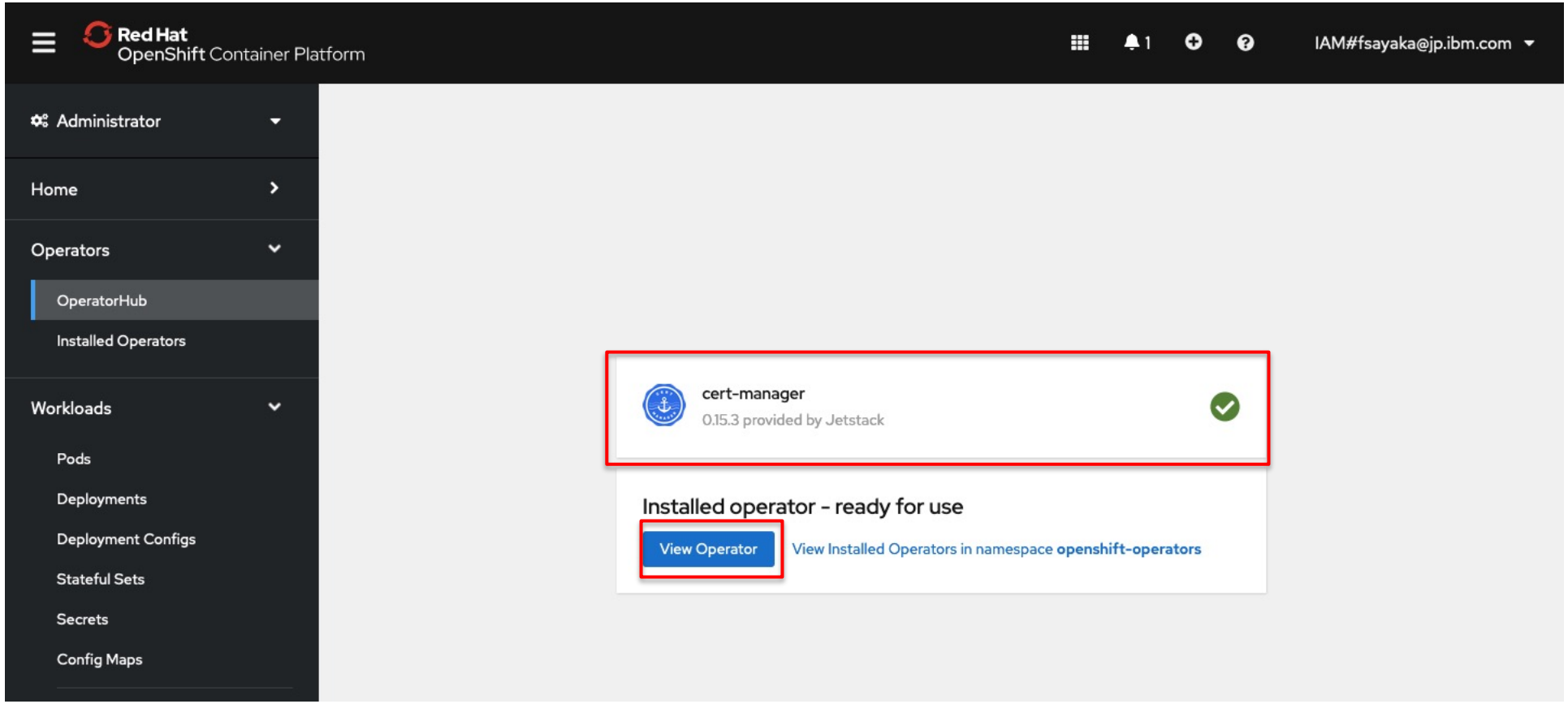
2.1. OperatorによるGitLabのインストール – 4.Cert Manager の導入

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



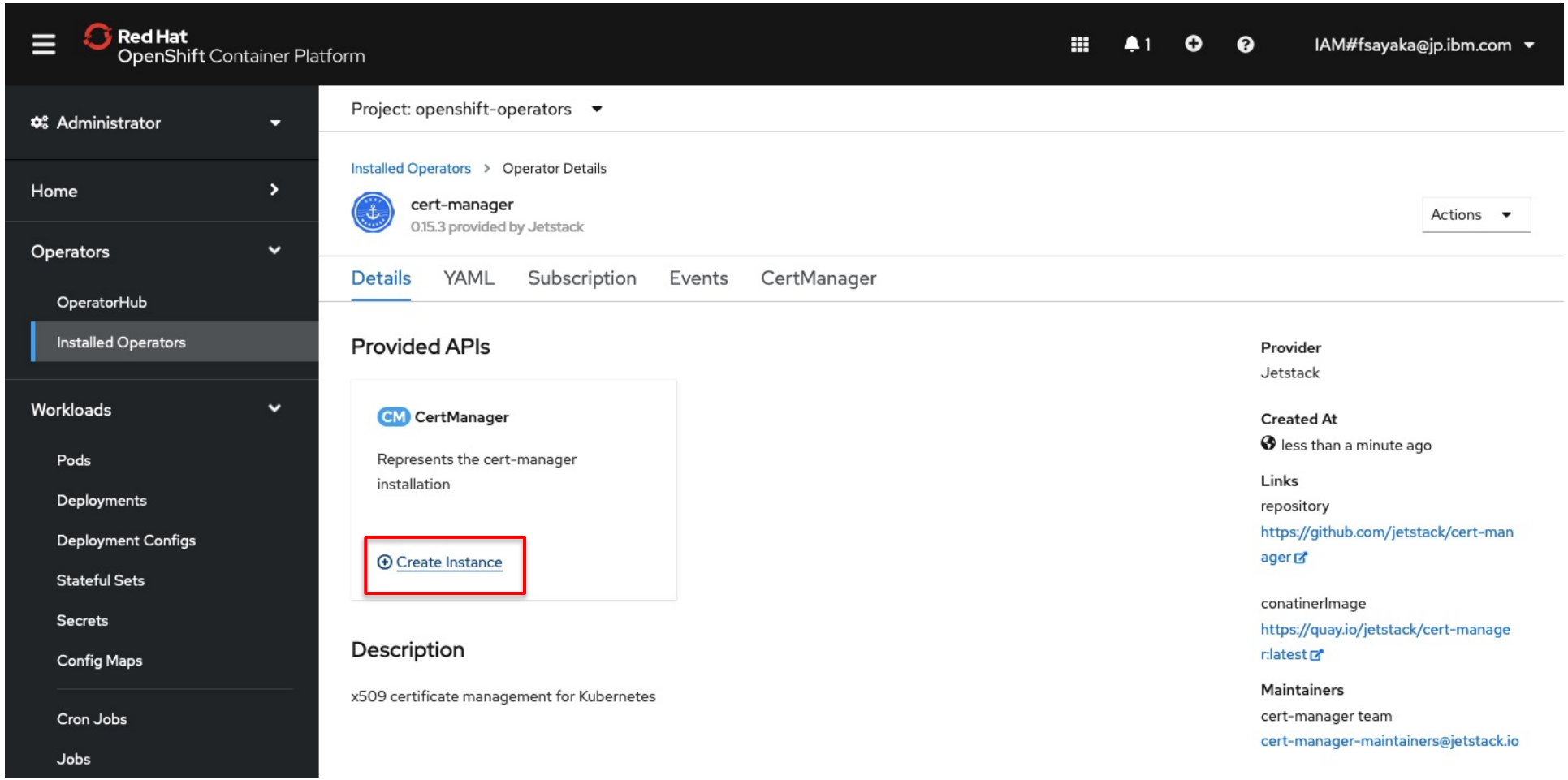
2.1. OperatorによるGitLabのインストール – 4.Cert Manager の導入

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



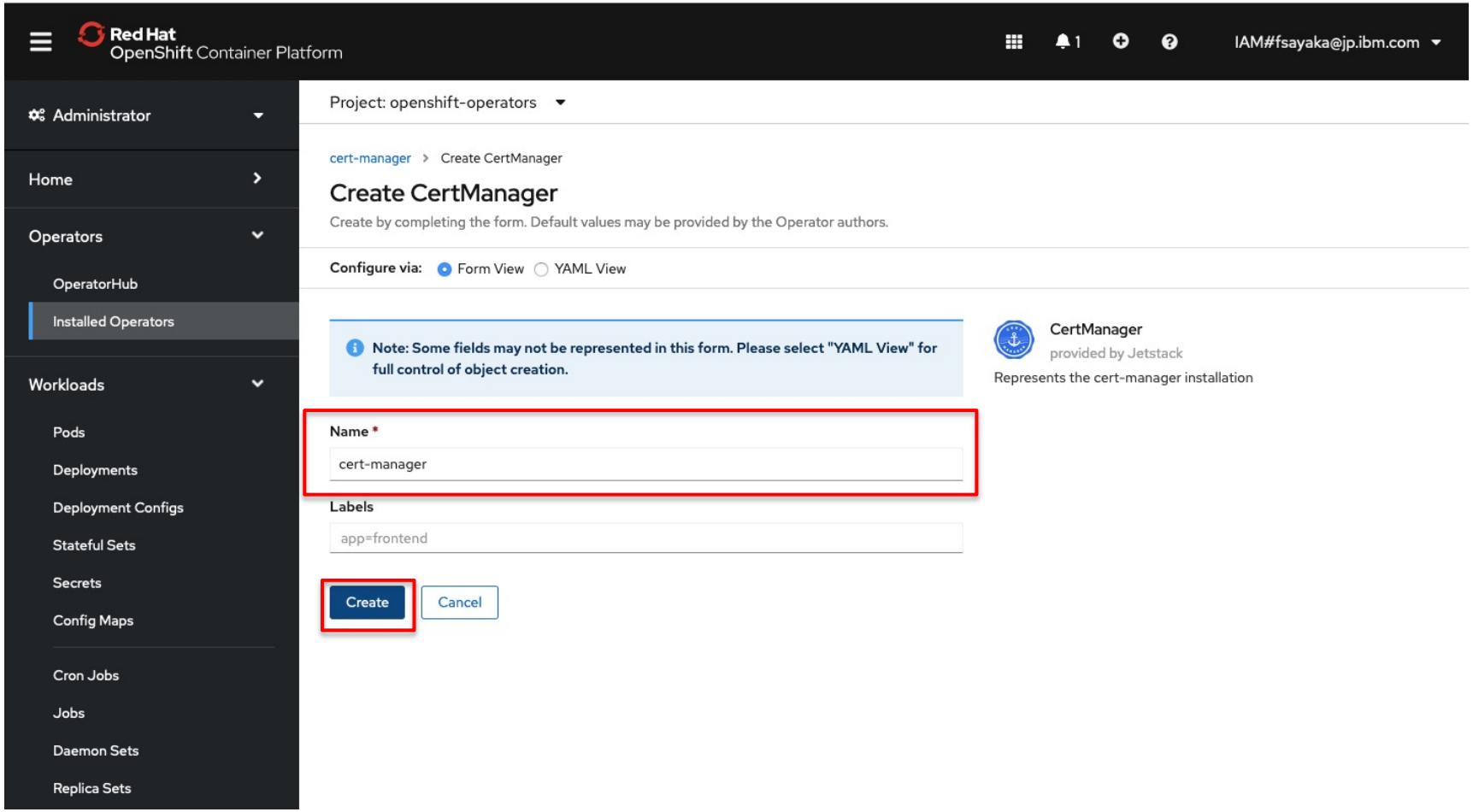
2.1. OperatorによるGitLabのインストール – 4.Cert Manager の導入

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



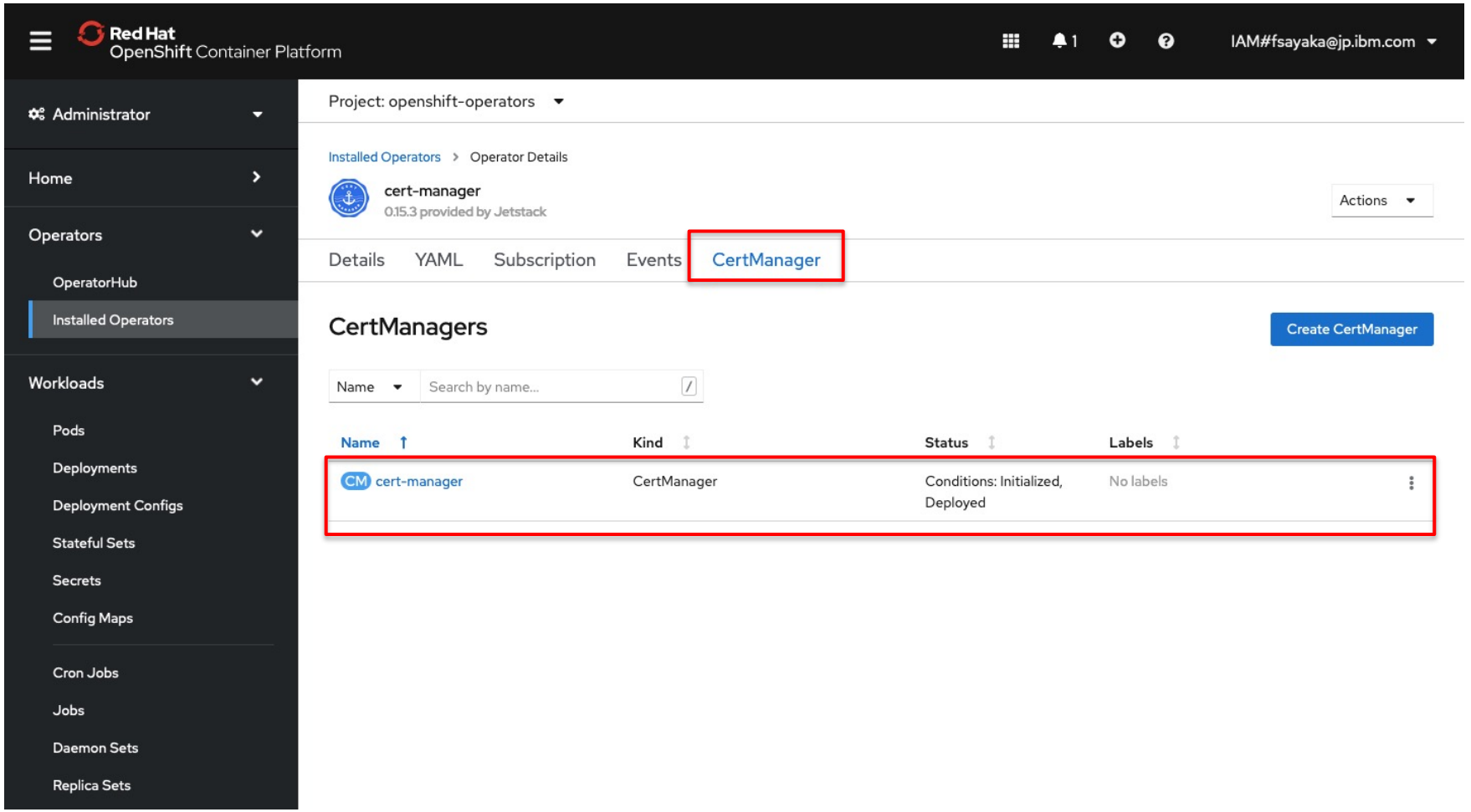
2.1. OperatorによるGitLabのインストール – 4.Cert Manager の導入

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



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



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リポジトリーの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リポジトリーのclone

1. Macの場合ターミナル、Windowsの場合はPowerShellを開く
2. Git リポジトリーを作成するディレクトリーへ移動（cd）する

3. 以下のコマンドを実行してGit operatorリポジトリーをcloneする

```
cd <Gitリポジトリー作成ディレクトリ>
```

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

4. cloneしたリポジトリーのディレクトリーに移動する

```
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_VERSION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-mbknz      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                                                                                    PATH
SERVICES                          PORT           TERMINATION     WILDCARD
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   gitlab.mycluster-tok02-b-889642-d85540d5ac7eb4665421ec4a86b8d368-
0000.jp-tok.containers.appdomain.cloud     /               gitlab-webservice-default   http-
workhorse    edge/Redirect   None
gitlab-webservice-default-j92d9   gitlab.mycluster-tok02-b-889642-d85540d5ac7eb4665421ec4a86b8d368-
0000.jp-tok.containers.appdomain.cloud     /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

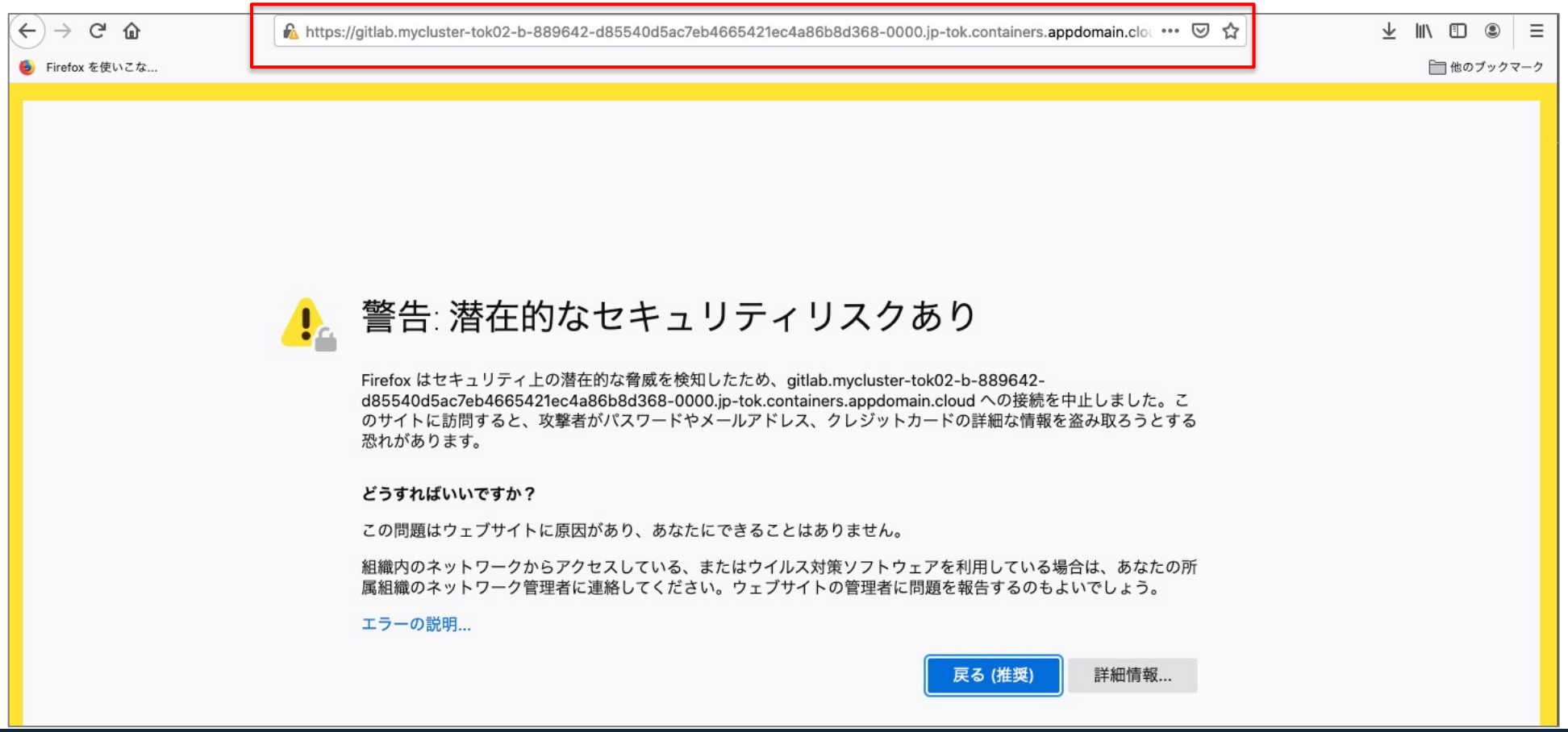
1. 以下のコマンドでrootユーザーのパスワードを確認する
   - <name>には「2.1. OperatorによるGitLabのインストール -5.OperatorとGitLabの導入」GitLab Custom Resource 定義ファイルの作成　で指定した metadata.name の値を指定する（本ガイドの例では gitlab）

```
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

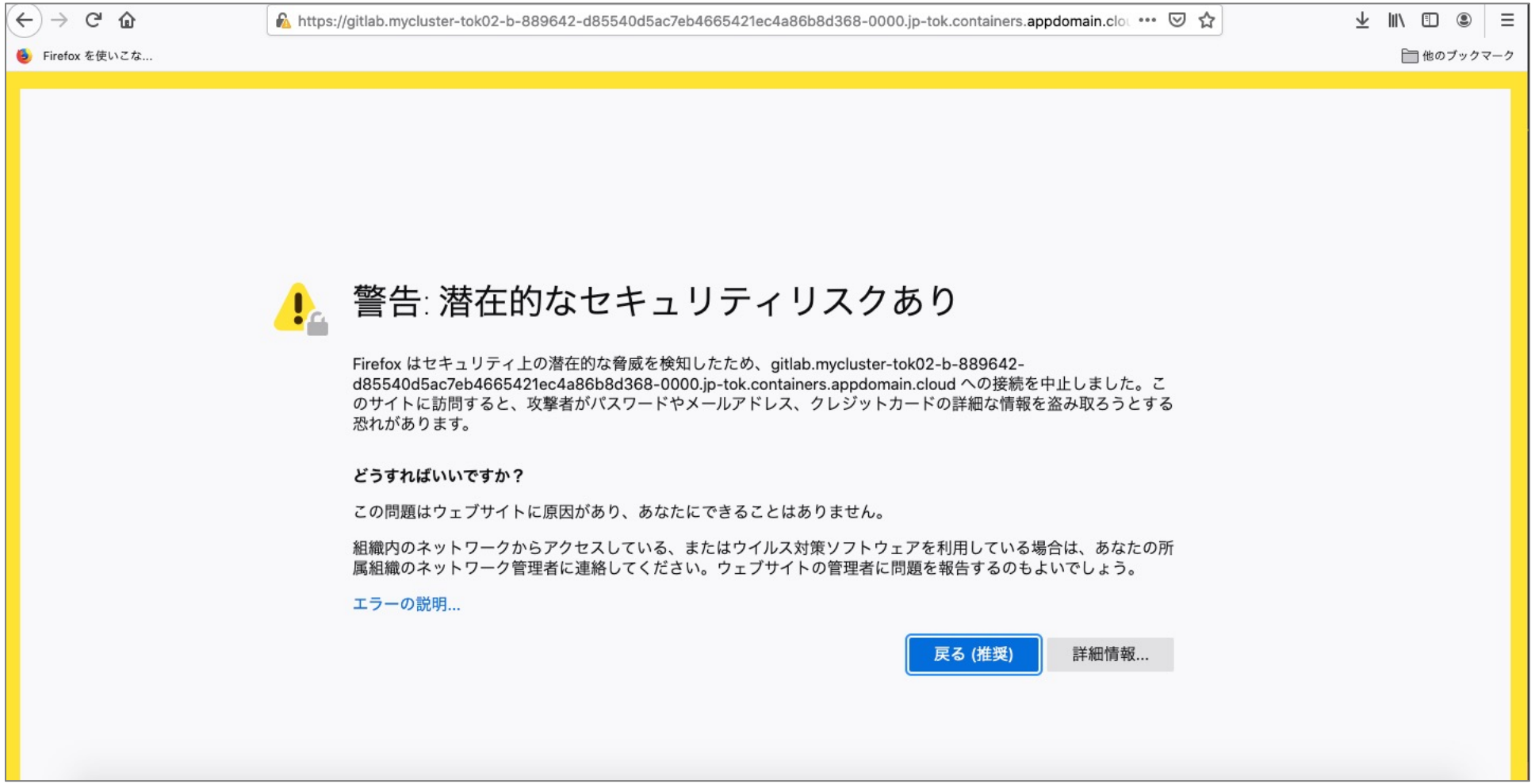
2.1. OperatorによるGitLabのインストール – 6. 接続確認

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



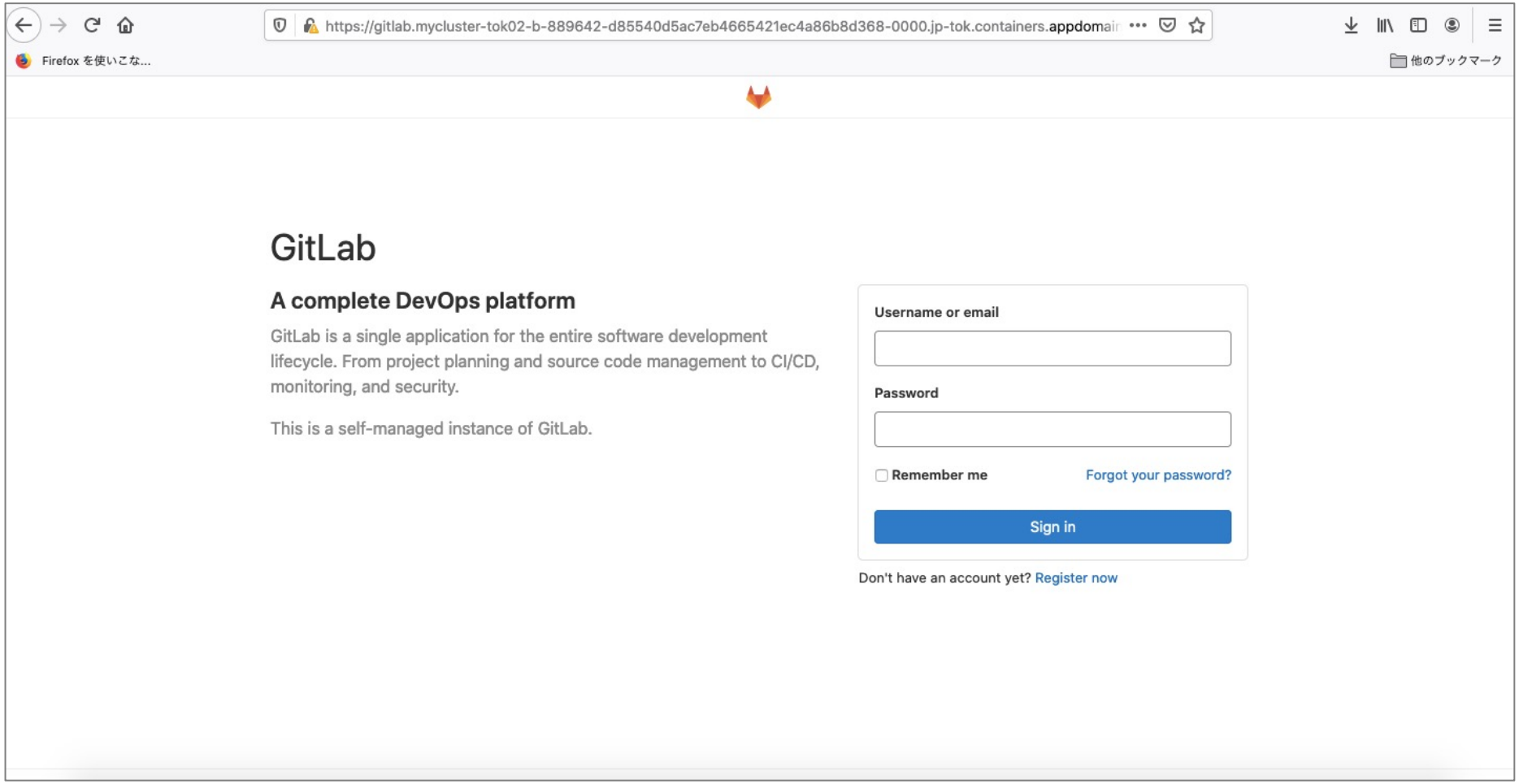
2.1. OperatorによるGitLabのインストール – 6. 接続確認

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



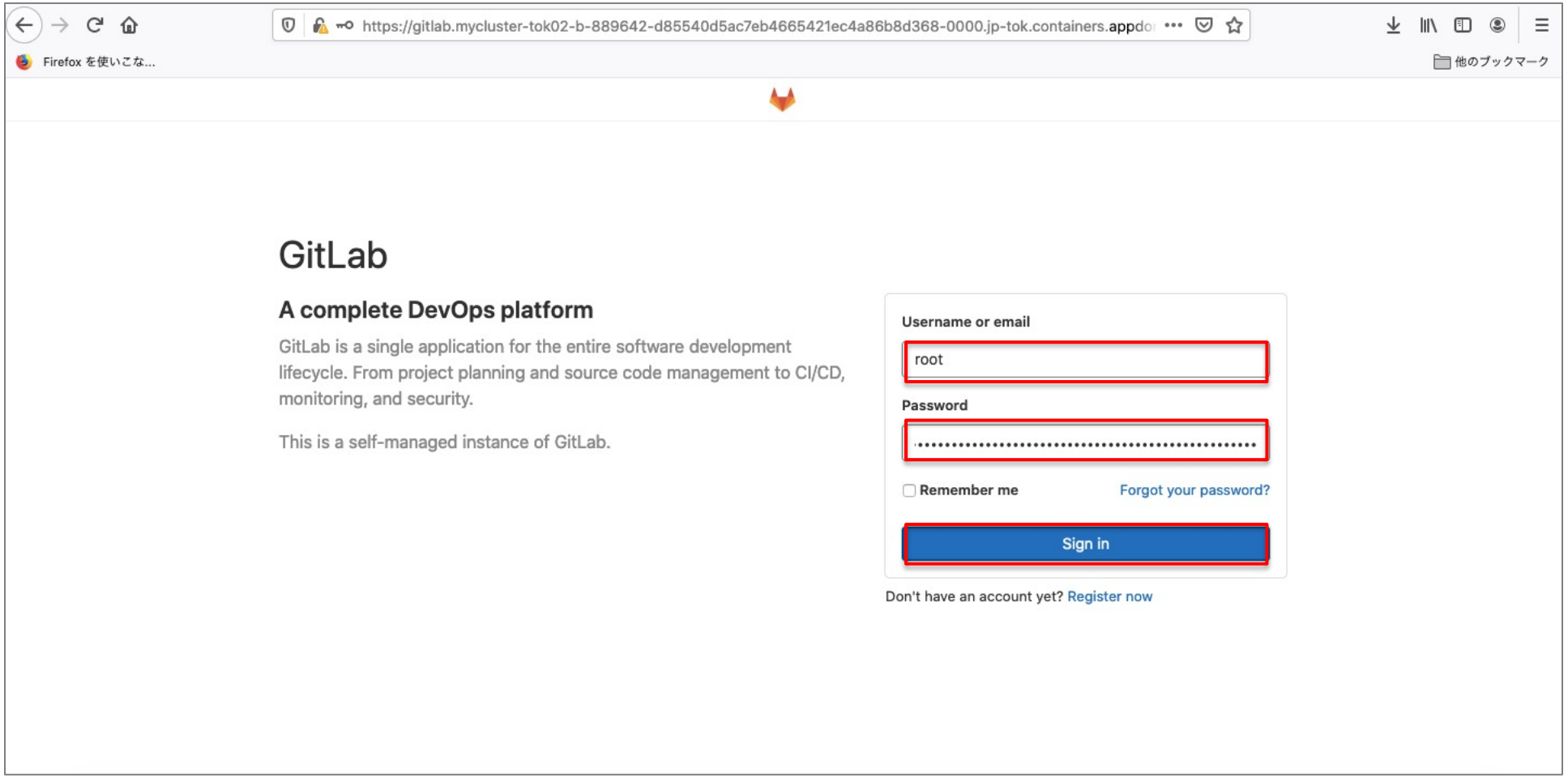
2.1. OperatorによるGitLabのインストール – 6. 接続確認

4. GitLabのログイン画面が表示されることを確認する



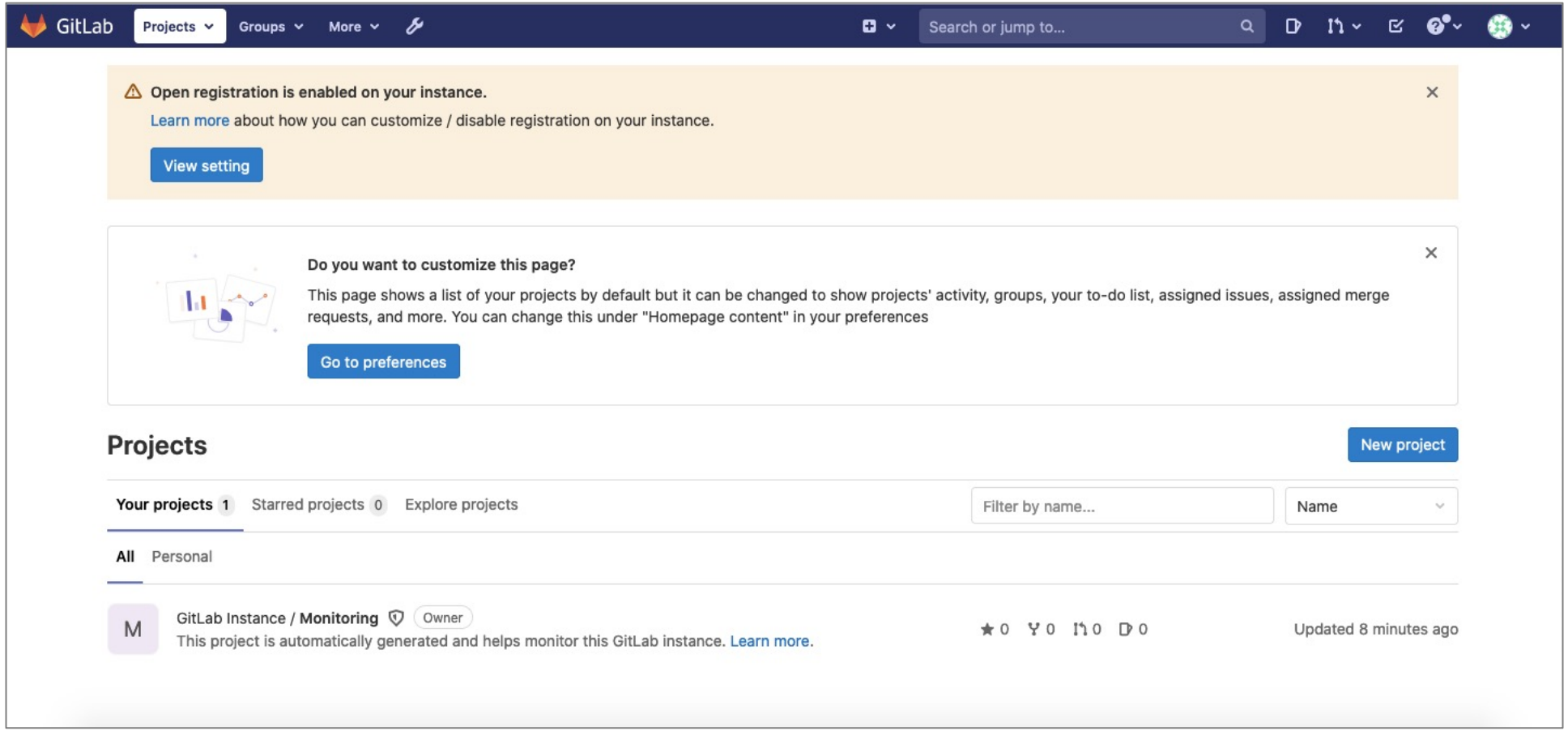
2.1. OperatorによるGitLabのインストール – 6. 接続確認

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



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

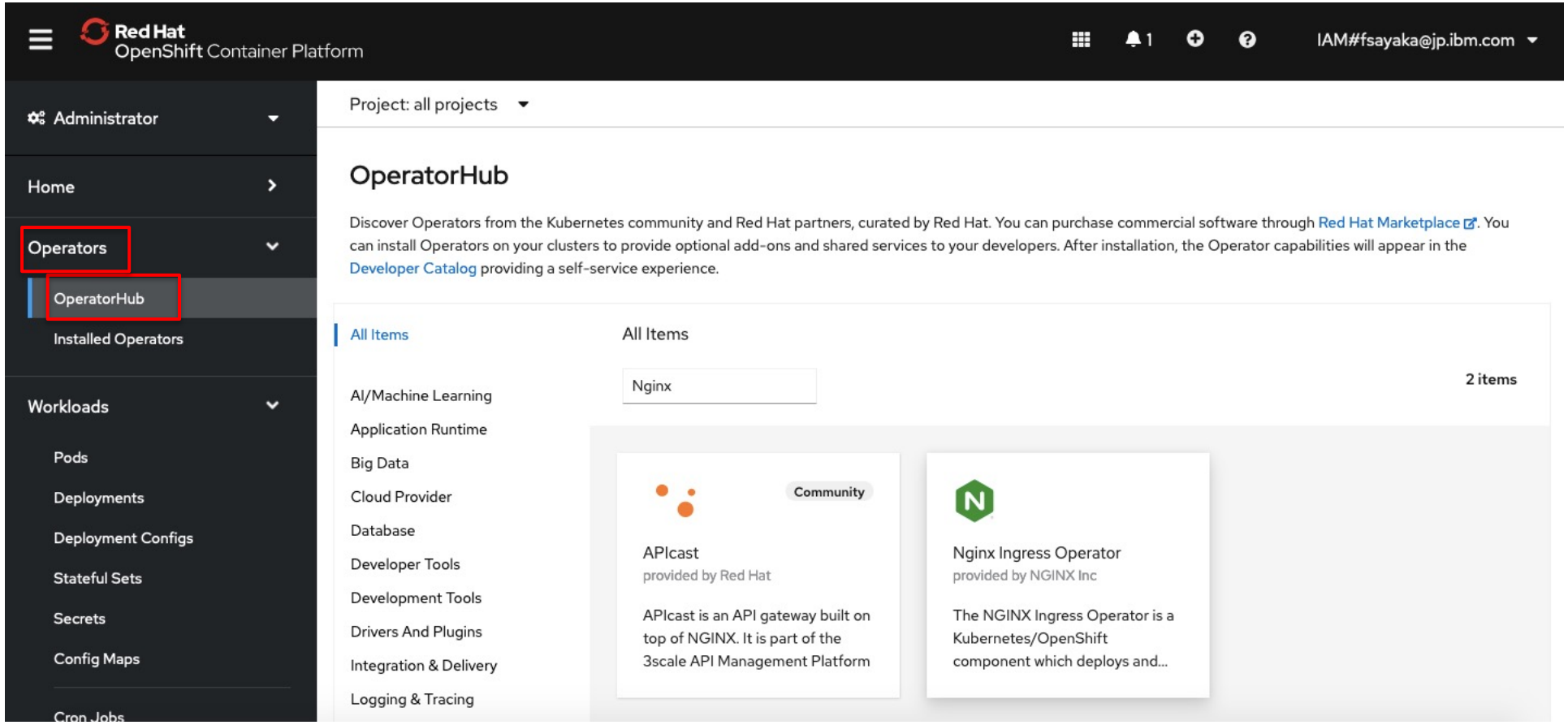
導入の流れは以下の通り

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

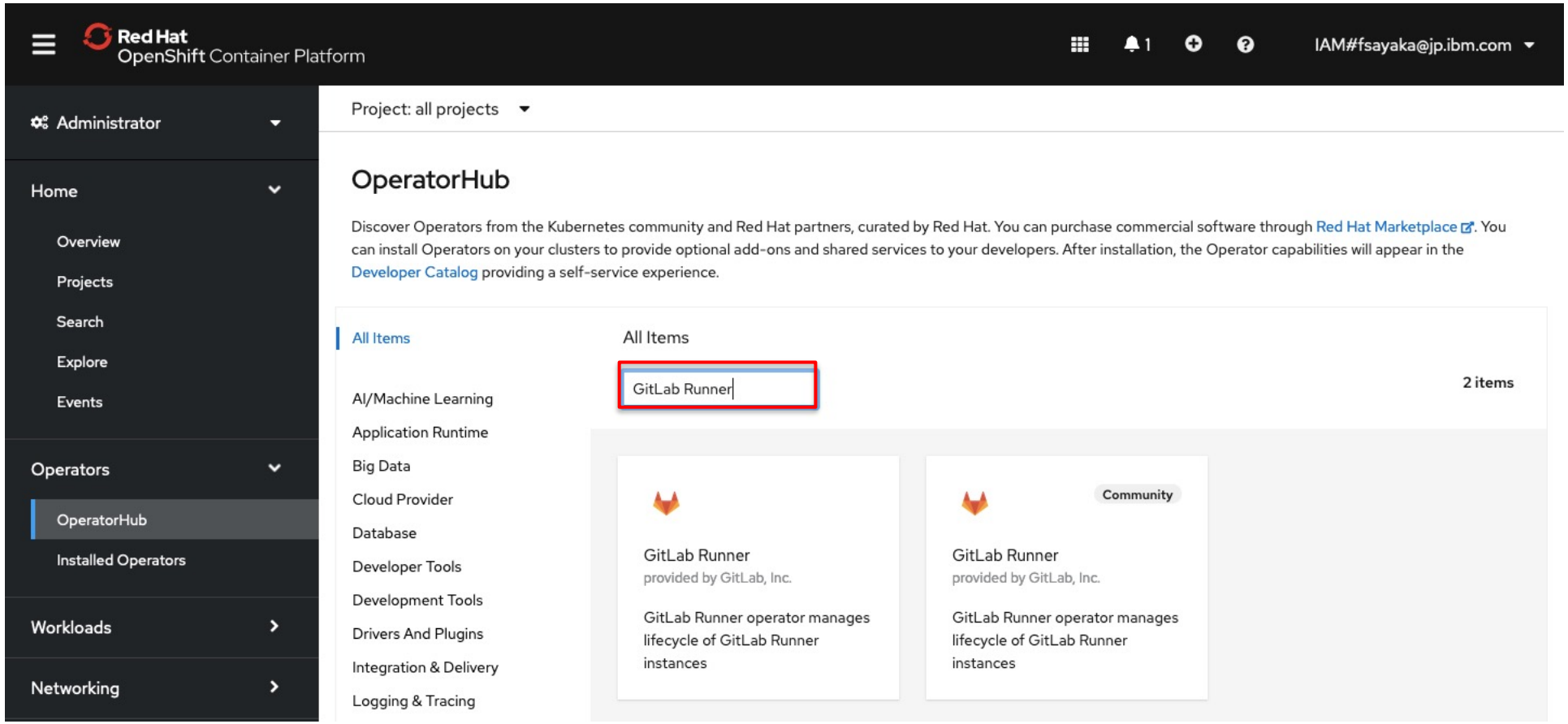
■ GitLab RunnerのOperatorを導入する

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



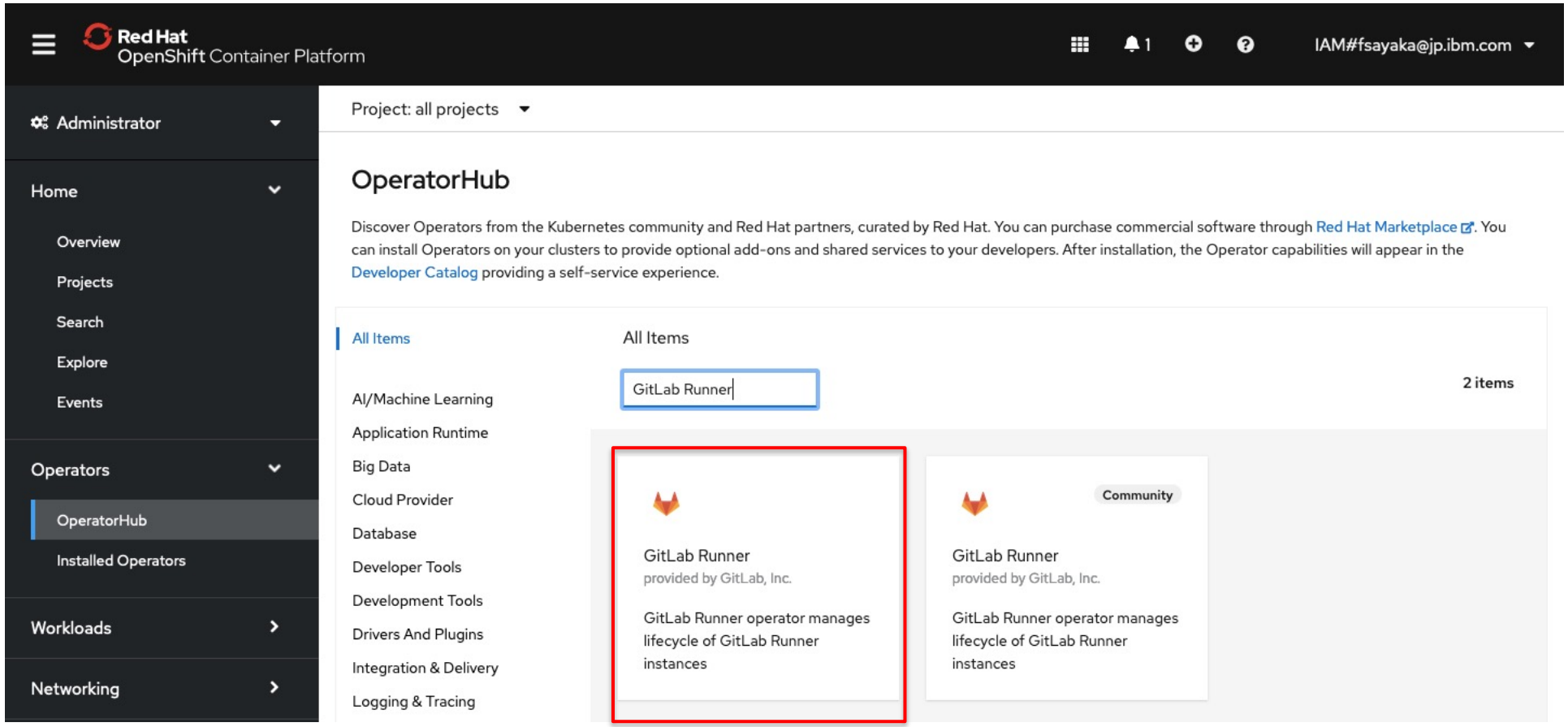
2.2 OperatorによるGitLab Runnerのインストール –1.GitLab Runner Operatorの導入

2. All itemsの下の欄に「GitLab Runner」と入力する



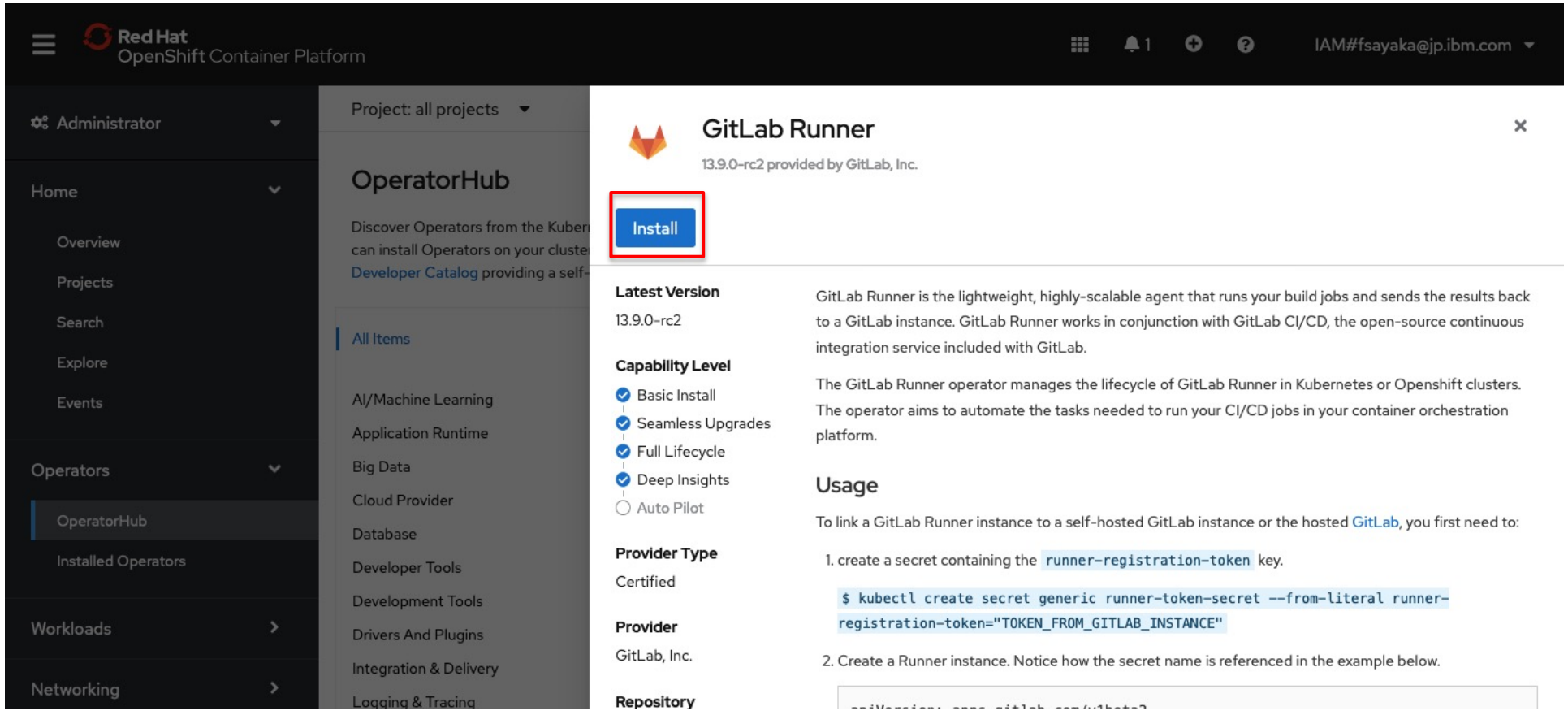
2.2 OperatorによるGitLab Runnerのインストール –1.GitLab Runner Operatorの導入

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



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

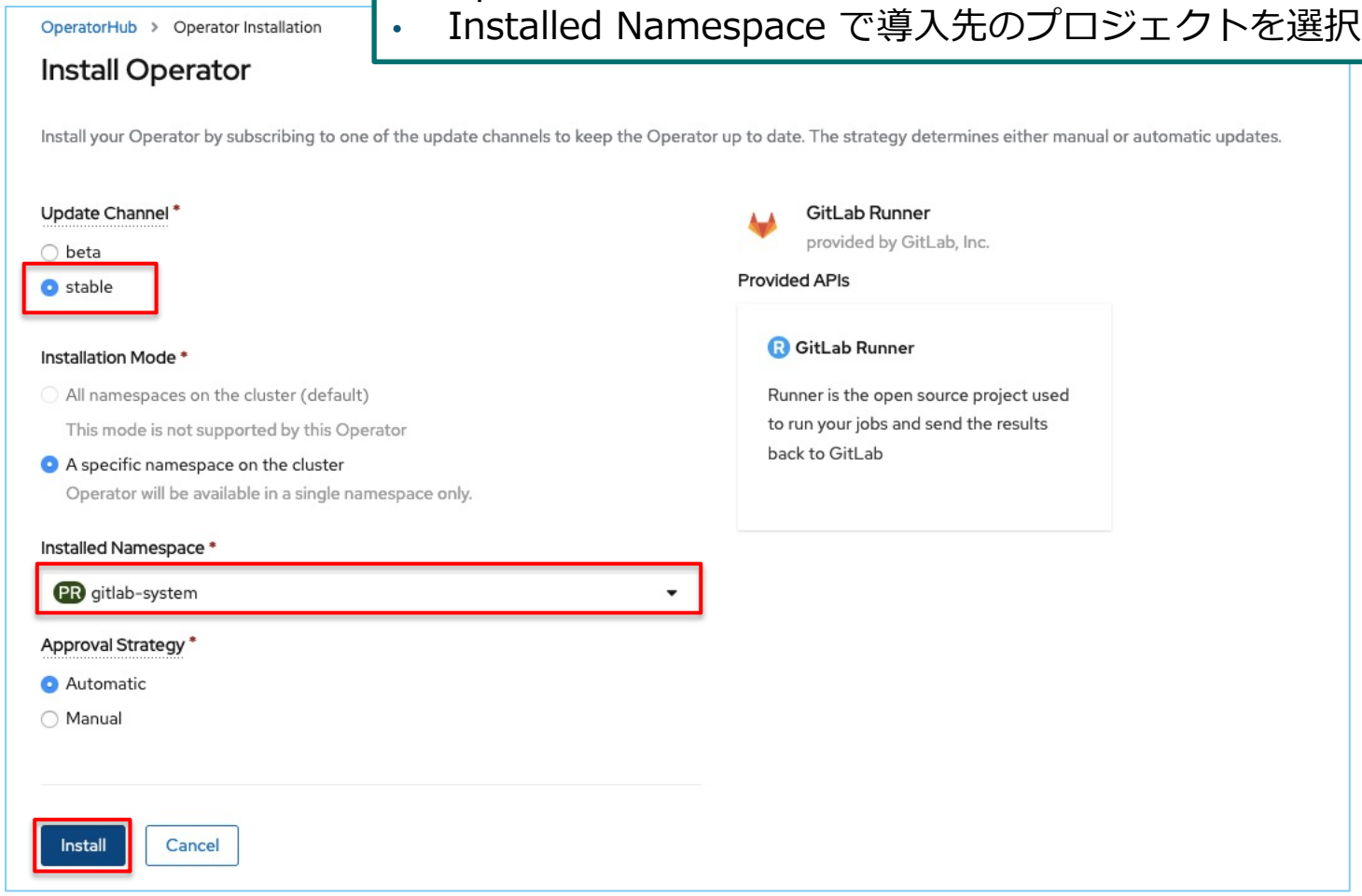
4.Operatorの詳細ページで「install」をクリックする



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

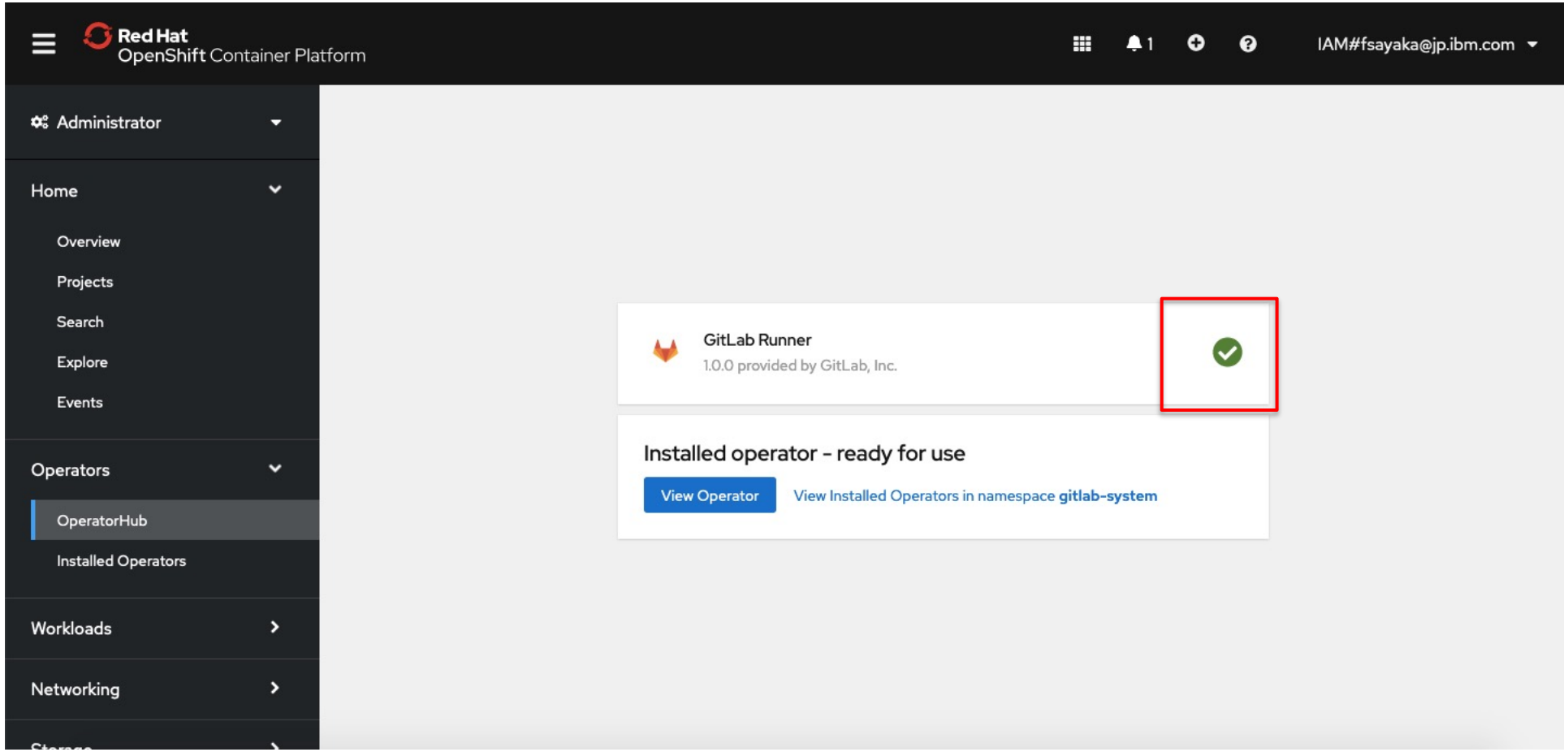
■ 5.Install Operatorページで以下のように設定し、「Install」をクリックする

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



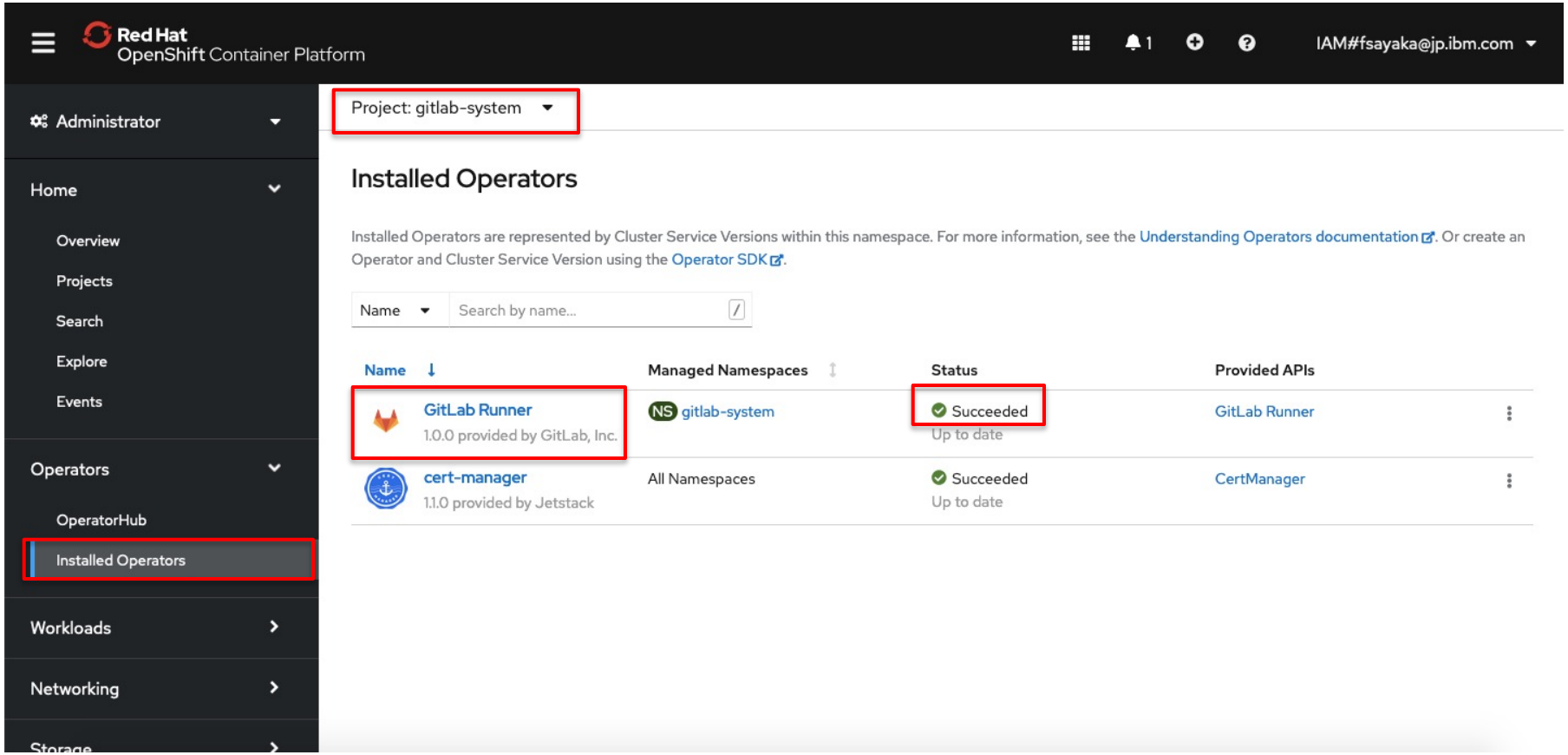
2.2 OperatorによるGitLab Runnerのインストール –1.GitLab Runner Operatorの導入

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



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

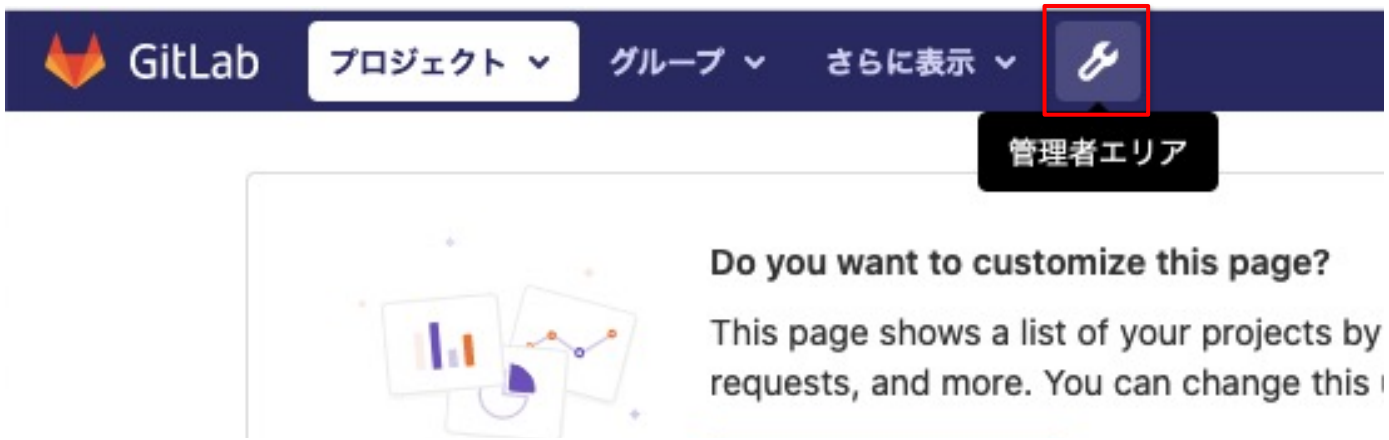
Runnerの種類によって確認方法が異なる

– 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

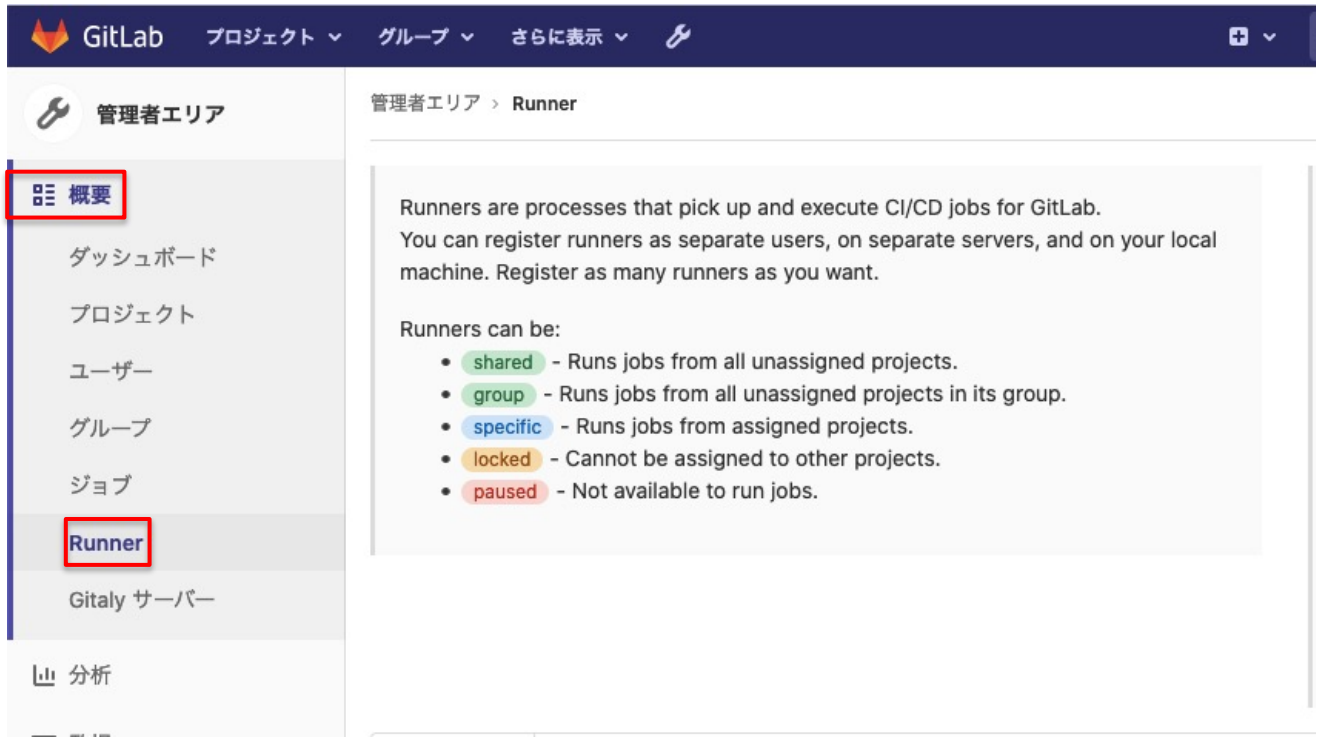
■ 共有Runner用の登録トークンとURLの確認

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



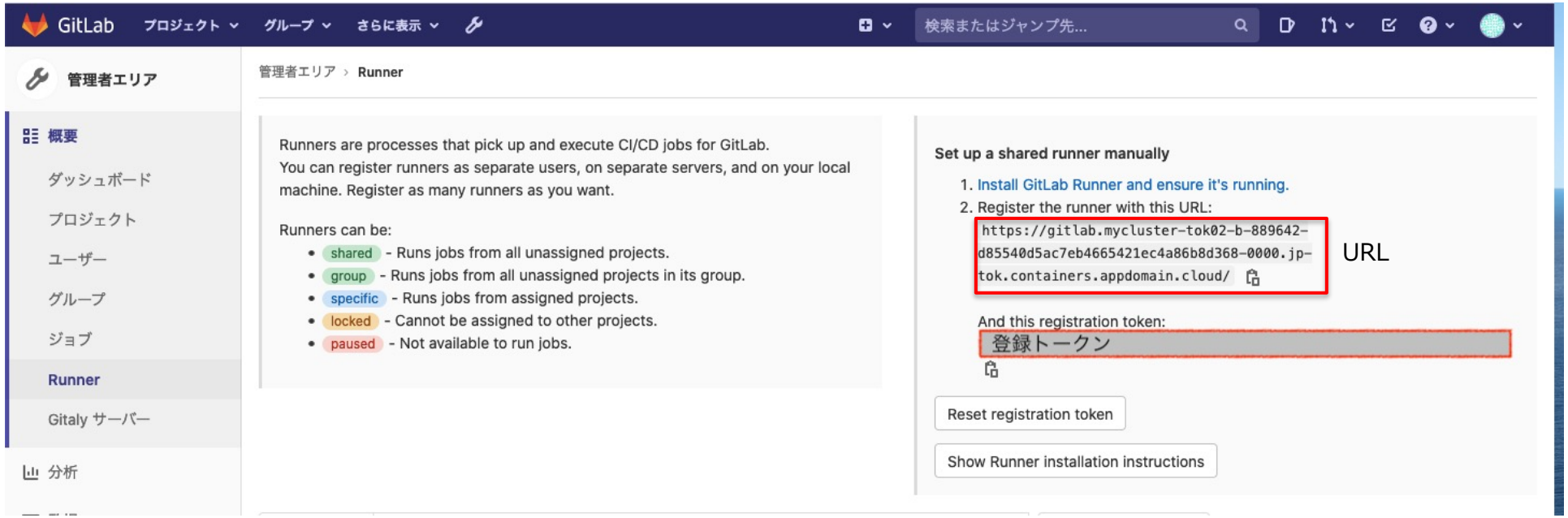
2.2 OperatorによるGitLab Runnerのインストール –2. GitLabRunner登録トークン、URLの確認とsecretの作成

2.管理者エリアの左側の「概要」の下の「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



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

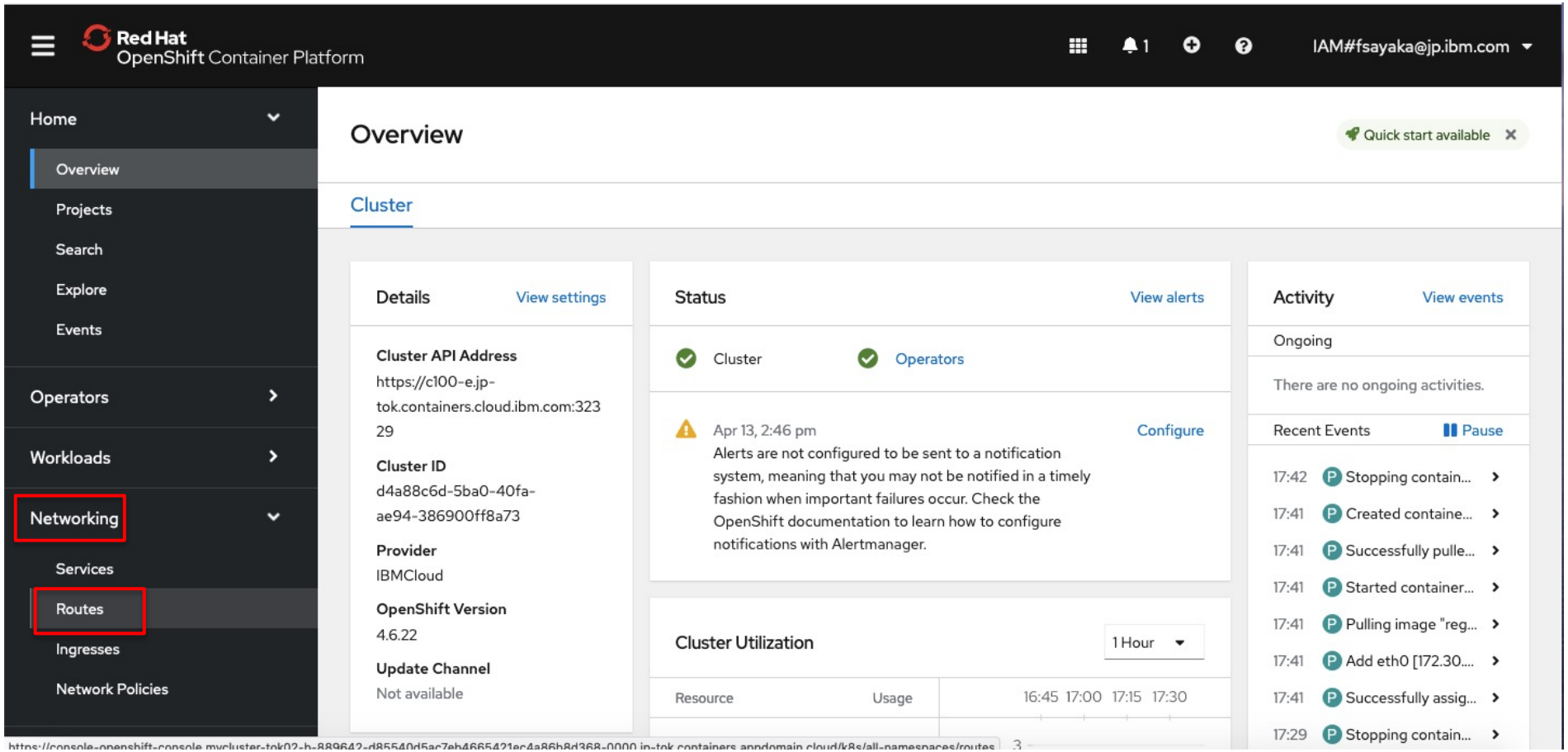
  参考：GItLabマニュアル Configure a custom TLS cert

  https://docs.gitlab.com/runner/configuration/configuring_runner_openshift.html#configure-a-custom-tls-cert

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

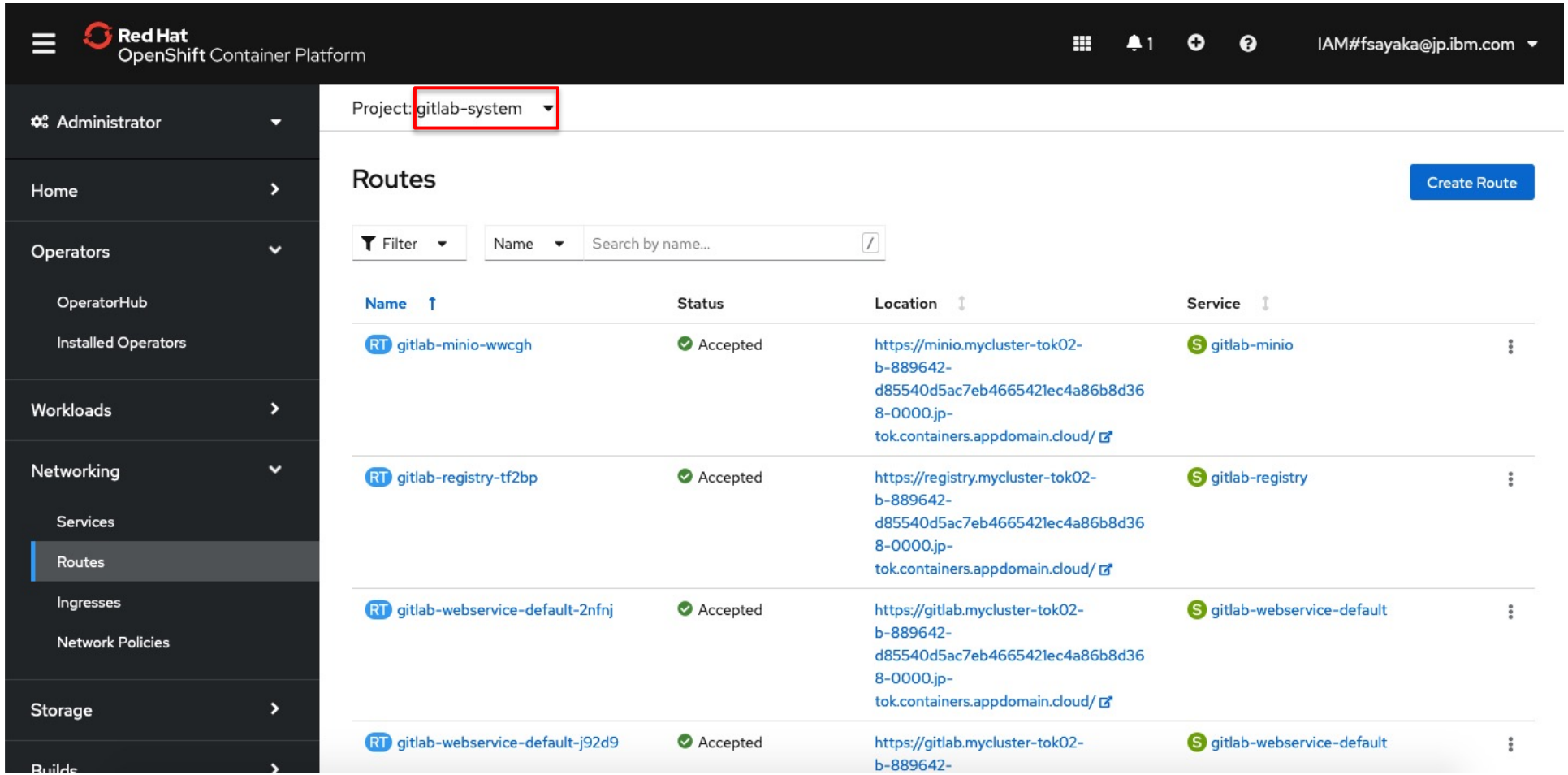
1. OpenShiftコンソールにログインし、左側のメニューから「Networking」を展開し「Routes」をクリックする



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

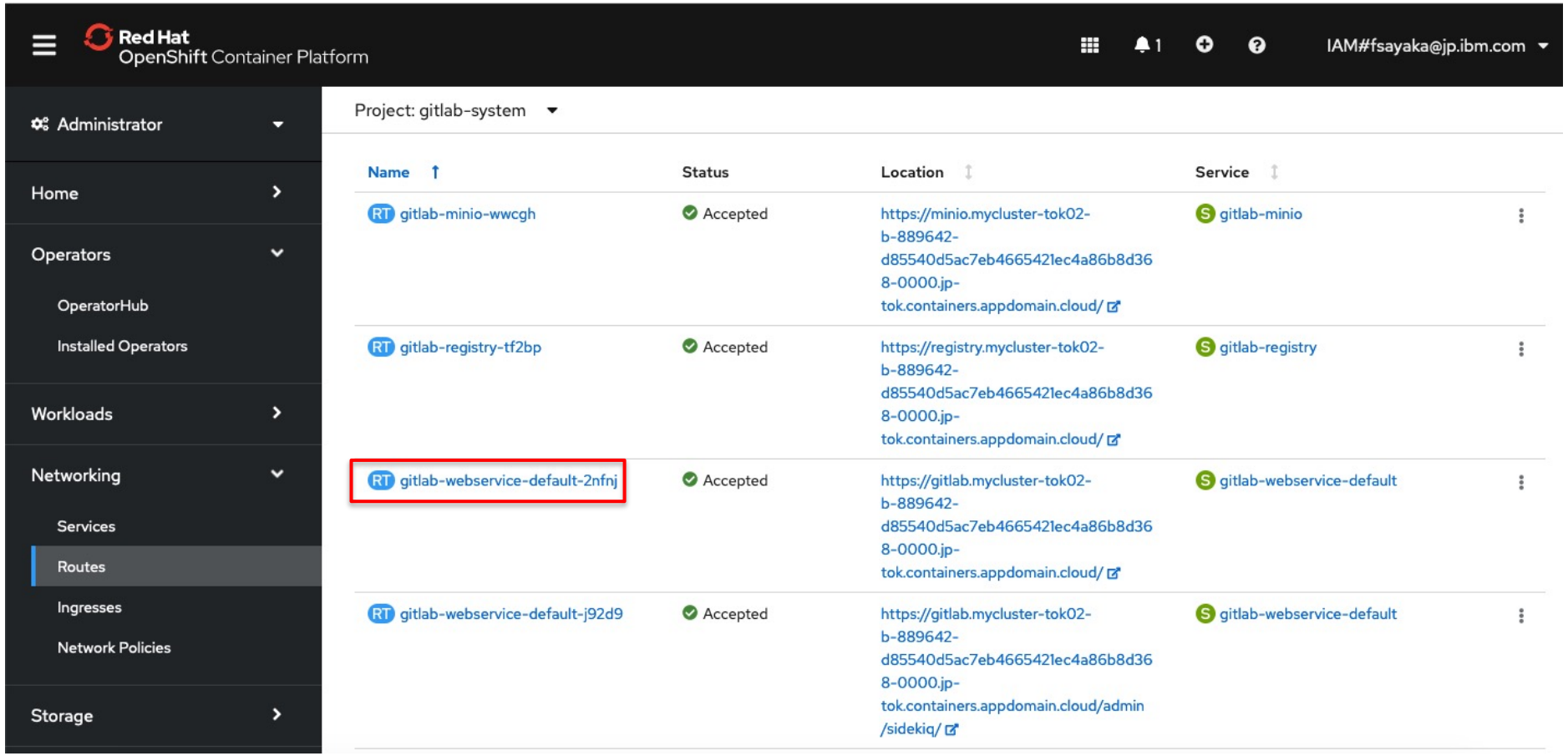
2. Projectのリストから「gitlab-system」を選択する



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

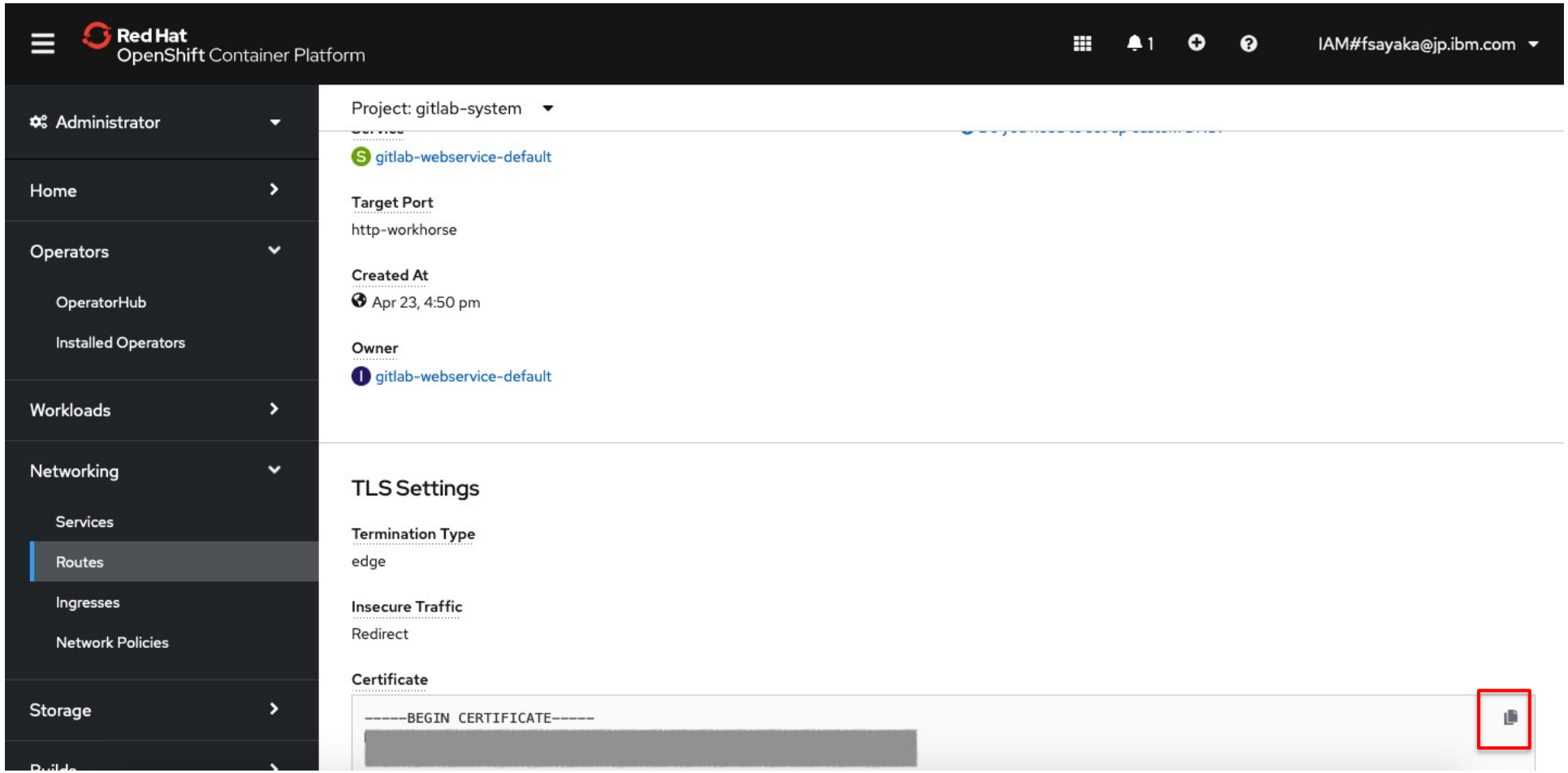
3. Name欄が「gitlab-web-service-default」で始まり、Locationが「/sidekiq/」で終わらない方のリンクをクリックして開く



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



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

5. 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..AkGA1UEBhMCR0Ix
        .....
        7vQMfXdGsRrXNGRGnX+vWDZ3/zWI0joDtCkNnqEpVn..HoX
        -----END CERTIFICATE-----
```

4. で確認したrouteのCertificateの値を記入

6.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

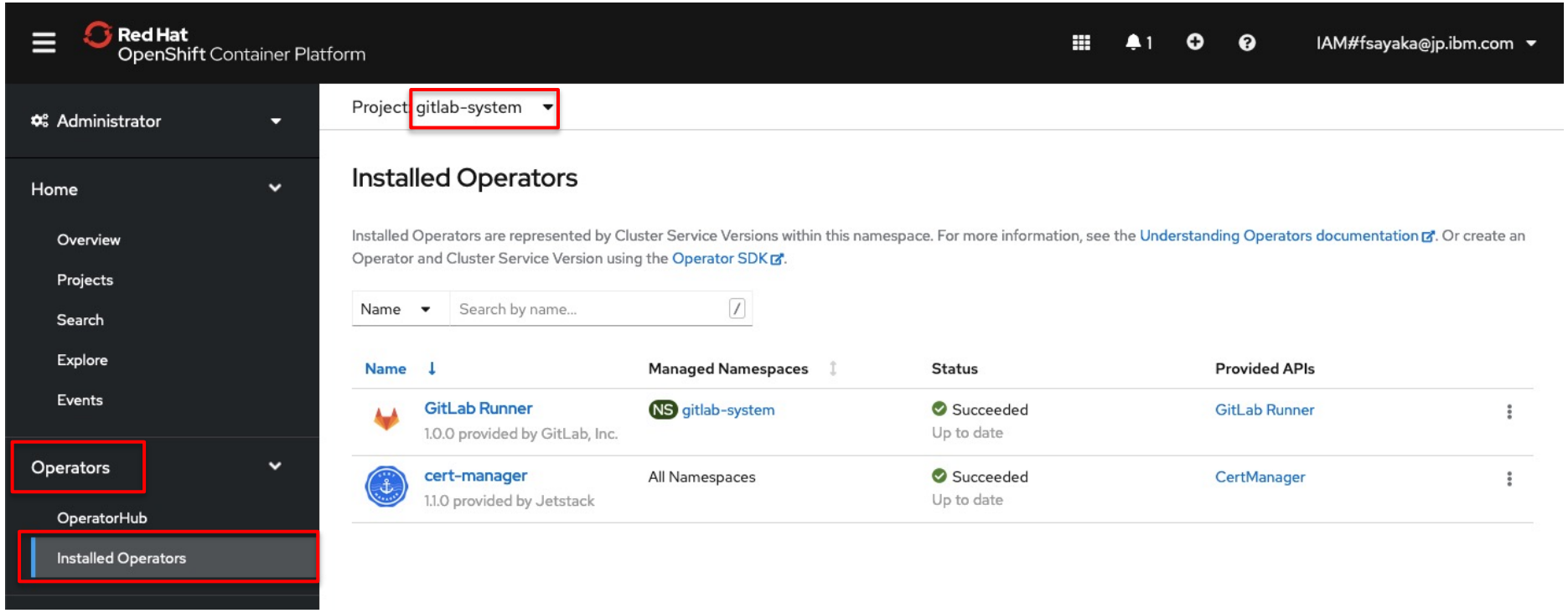
1. GitLab Runner Operatorの導入
2. GitLabRunner登録トークン、URLの確認とsecretの作成
3. （自己署名証明書を使用する場合）route証明書の確認とsecretの作成
4. **GitLab Runnerの導入**
5. 接続確認
6. 補足設定

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

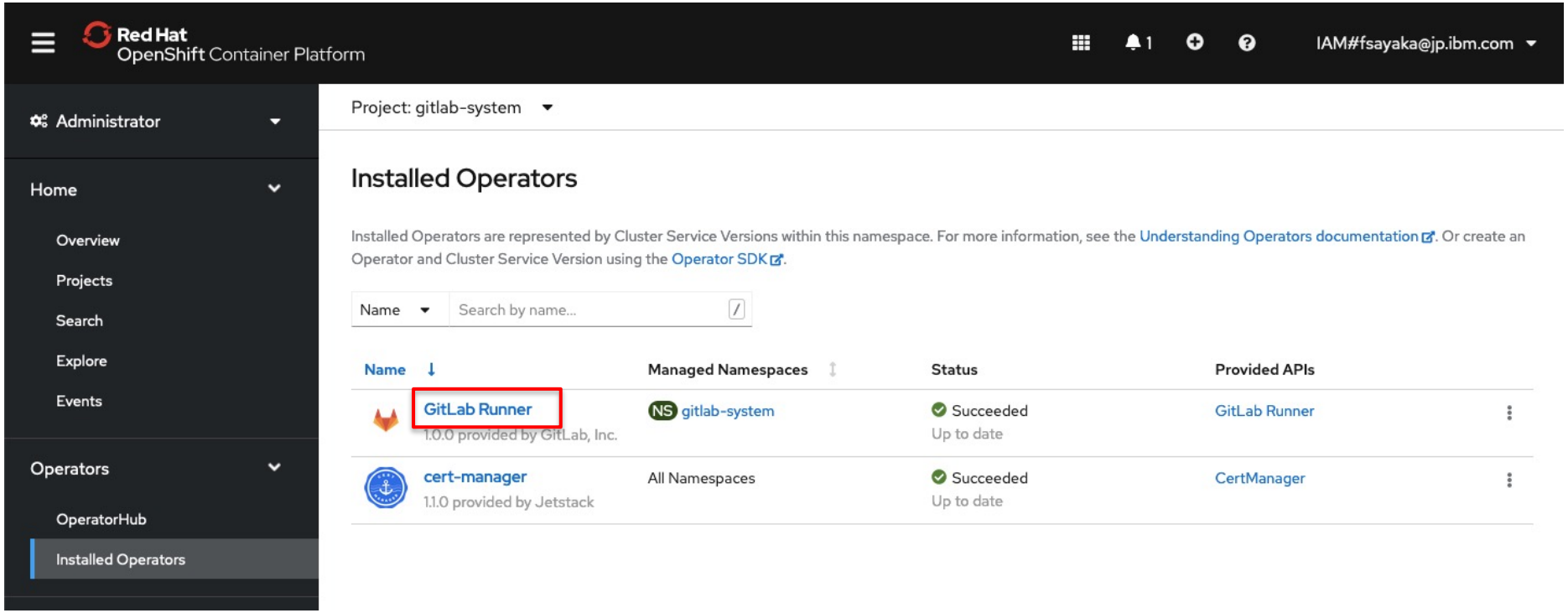
1.OpenShiftのWebコンソールに管理者としてログインする

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



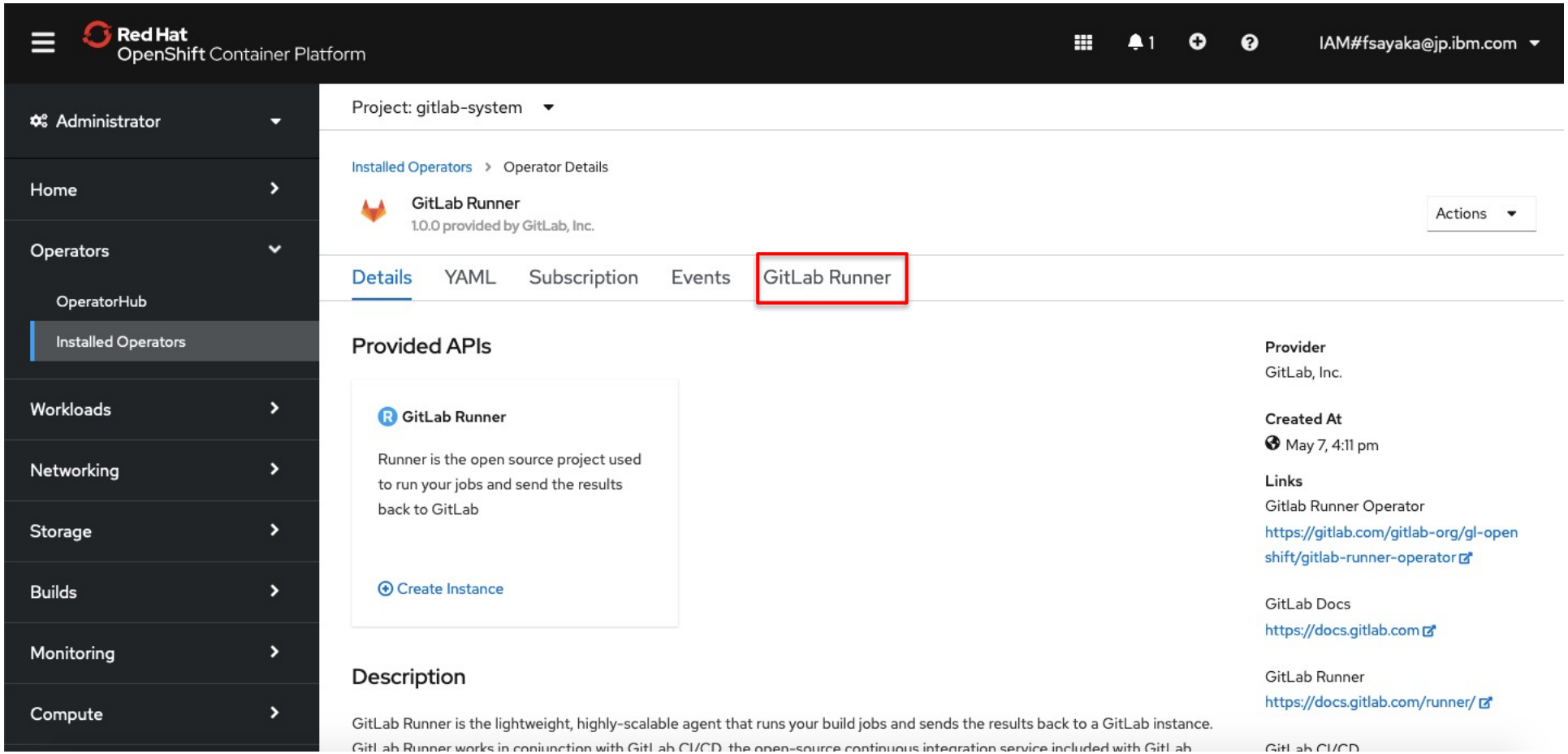
2.2 OperatorによるGitLab Runnerのインストール –4. GitLab Runnerの導入

3.表示された「GitLab Runner」をクリックする



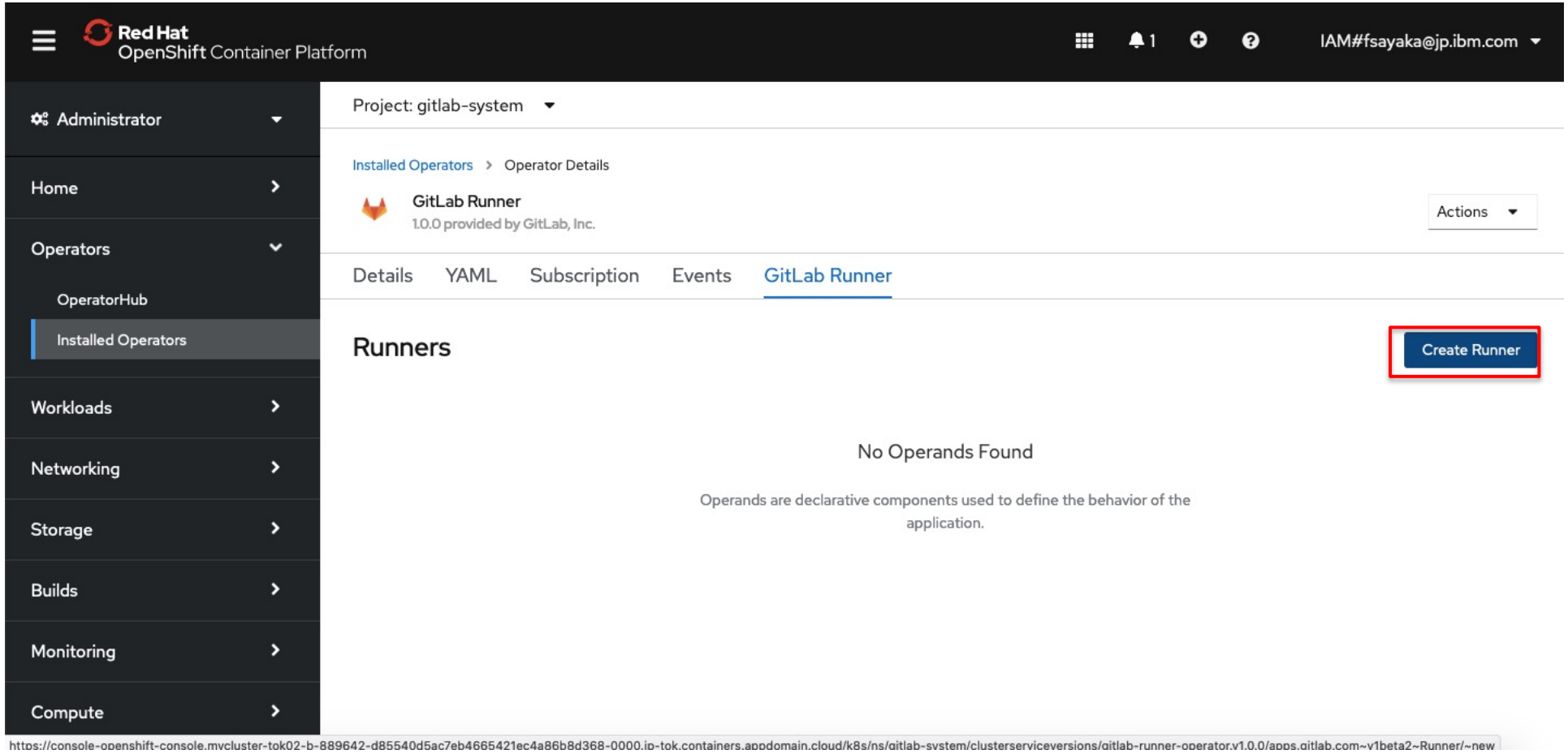
2.2 OperatorによるGitLab Runnerのインストール –4. GitLab Runnerの導入

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



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



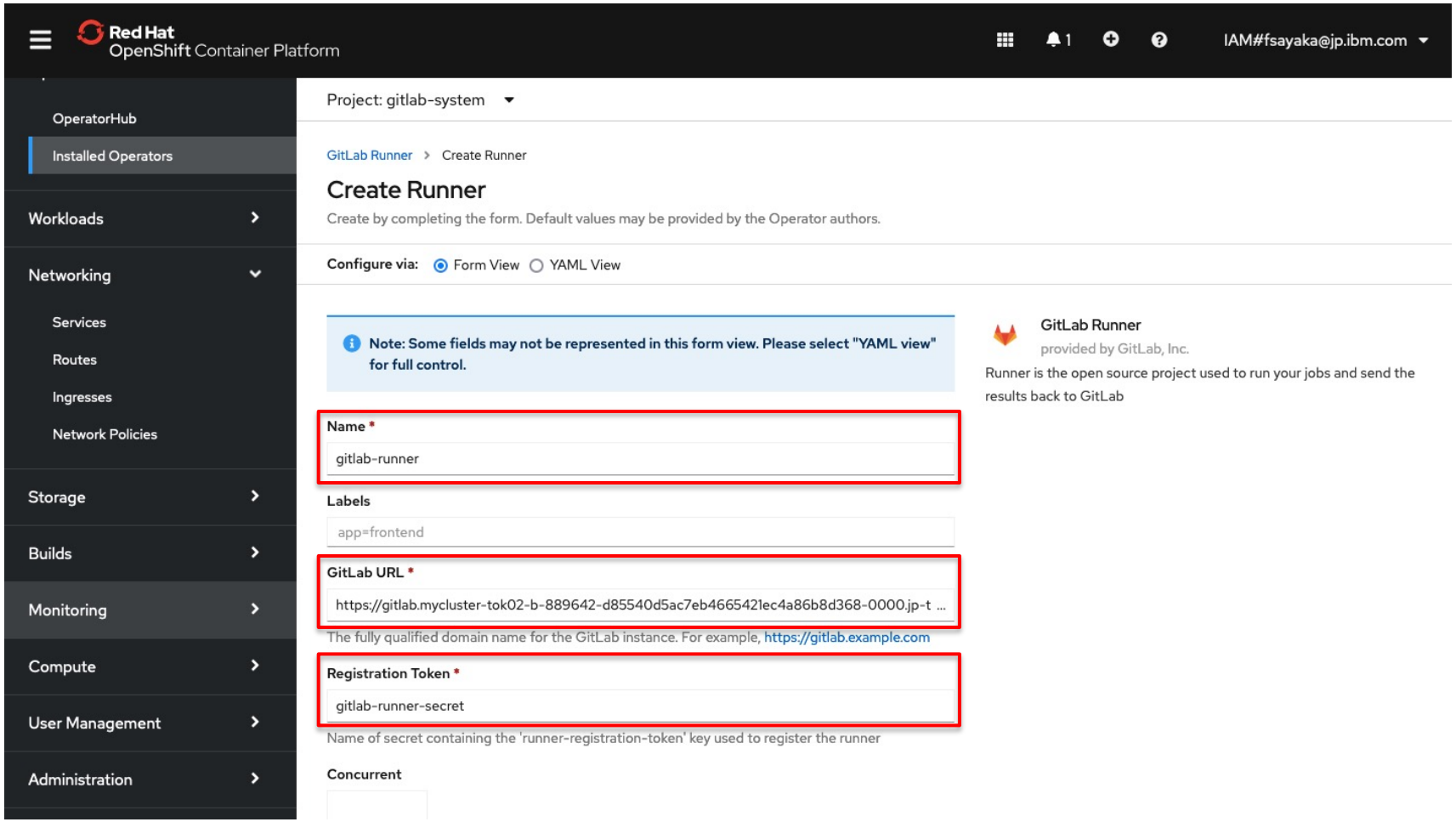
2.2 OperatorによるGitLab Runnerのインストール –4. GitLab Runnerの導入

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GitLabを参考にしている
- 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

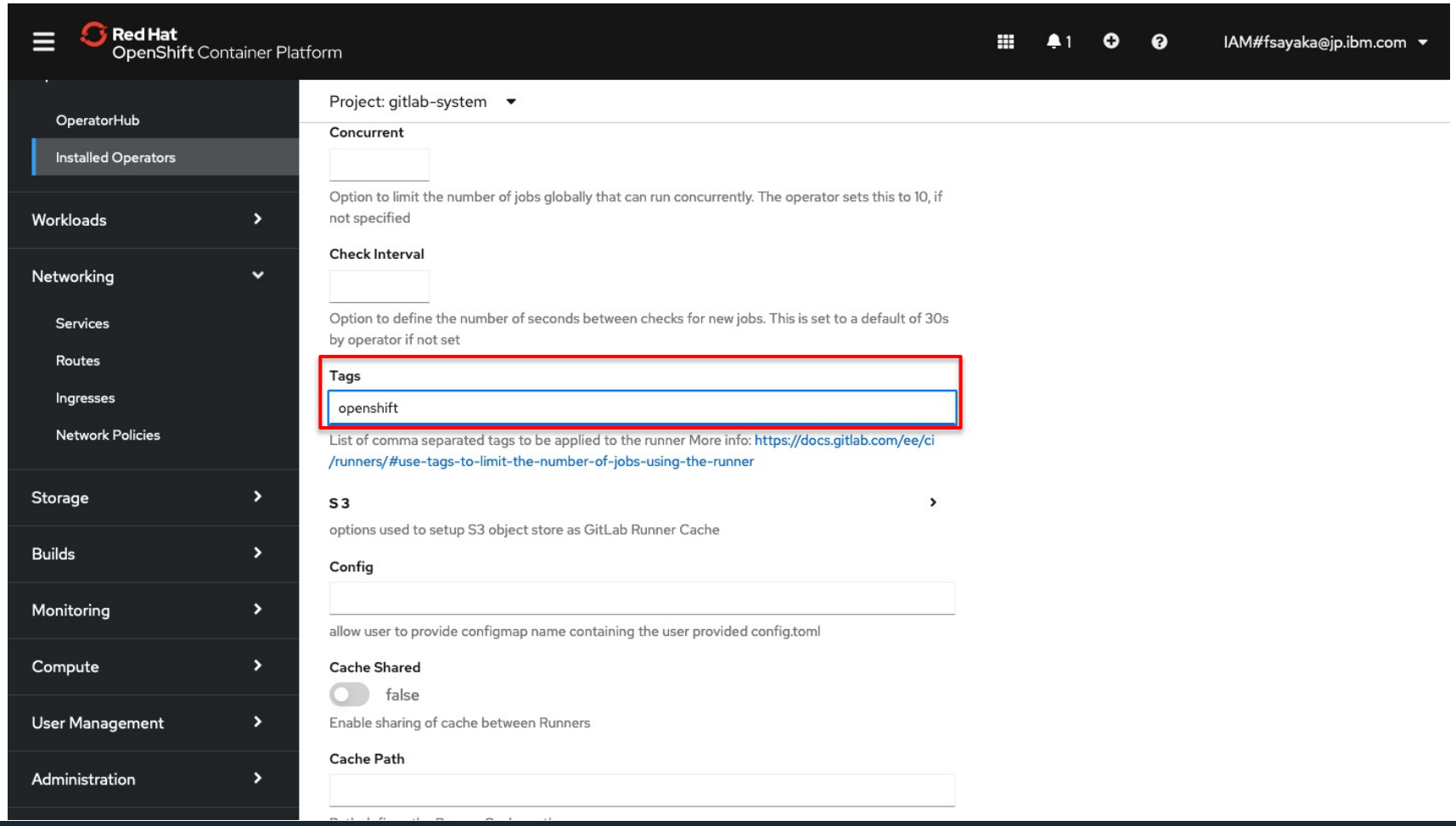
2.2 OperatorによるGitLab Runnerのインストール –4. GitLab Runnerの導入

6.表示された画面で以下の値を入力する
- 入力画面例



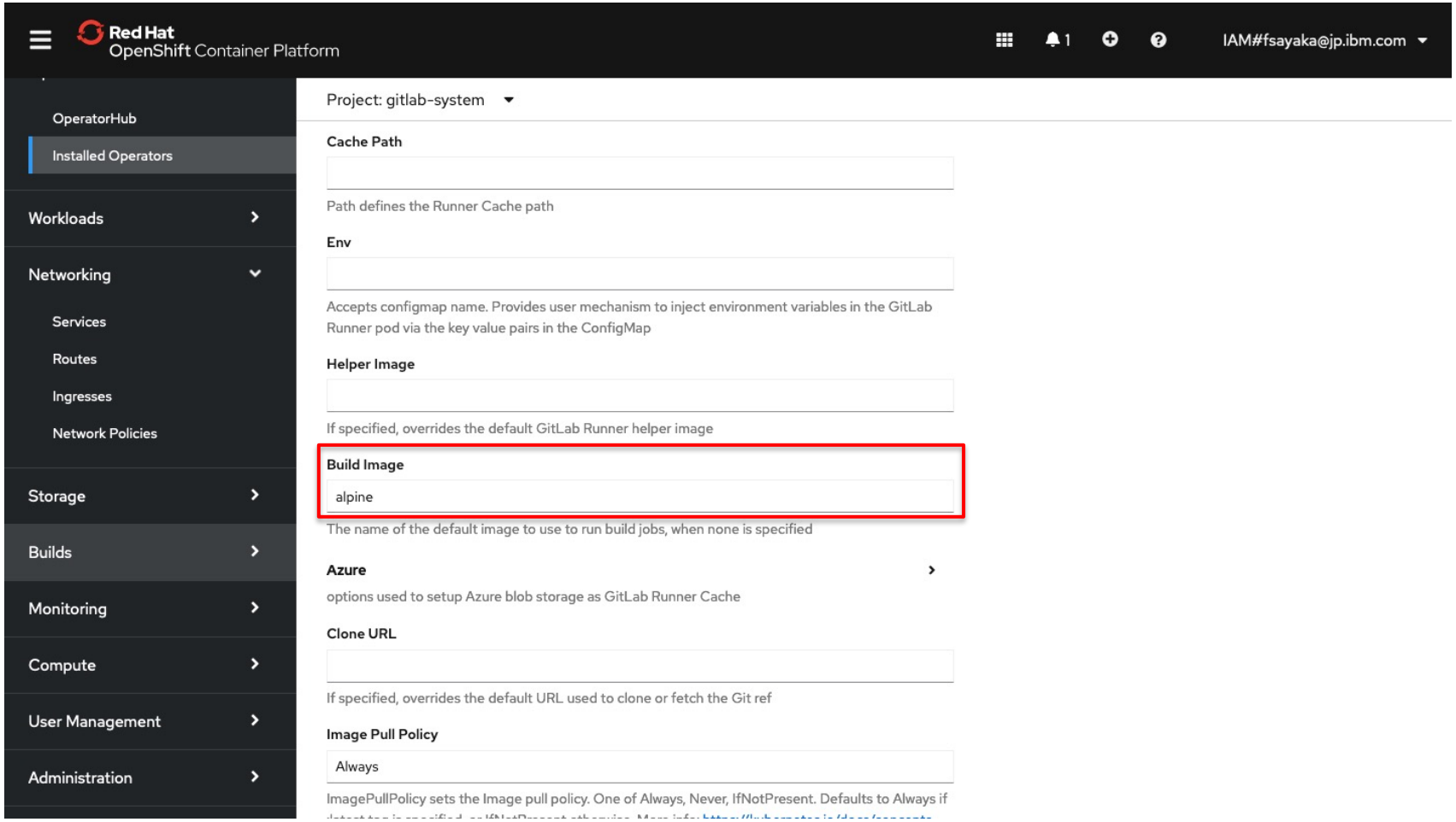
2.2 OperatorによるGitLab Runnerのインストール –4. GitLab Runnerの導入

6.表示された画面で以下の値を入力する
- 入力画面例



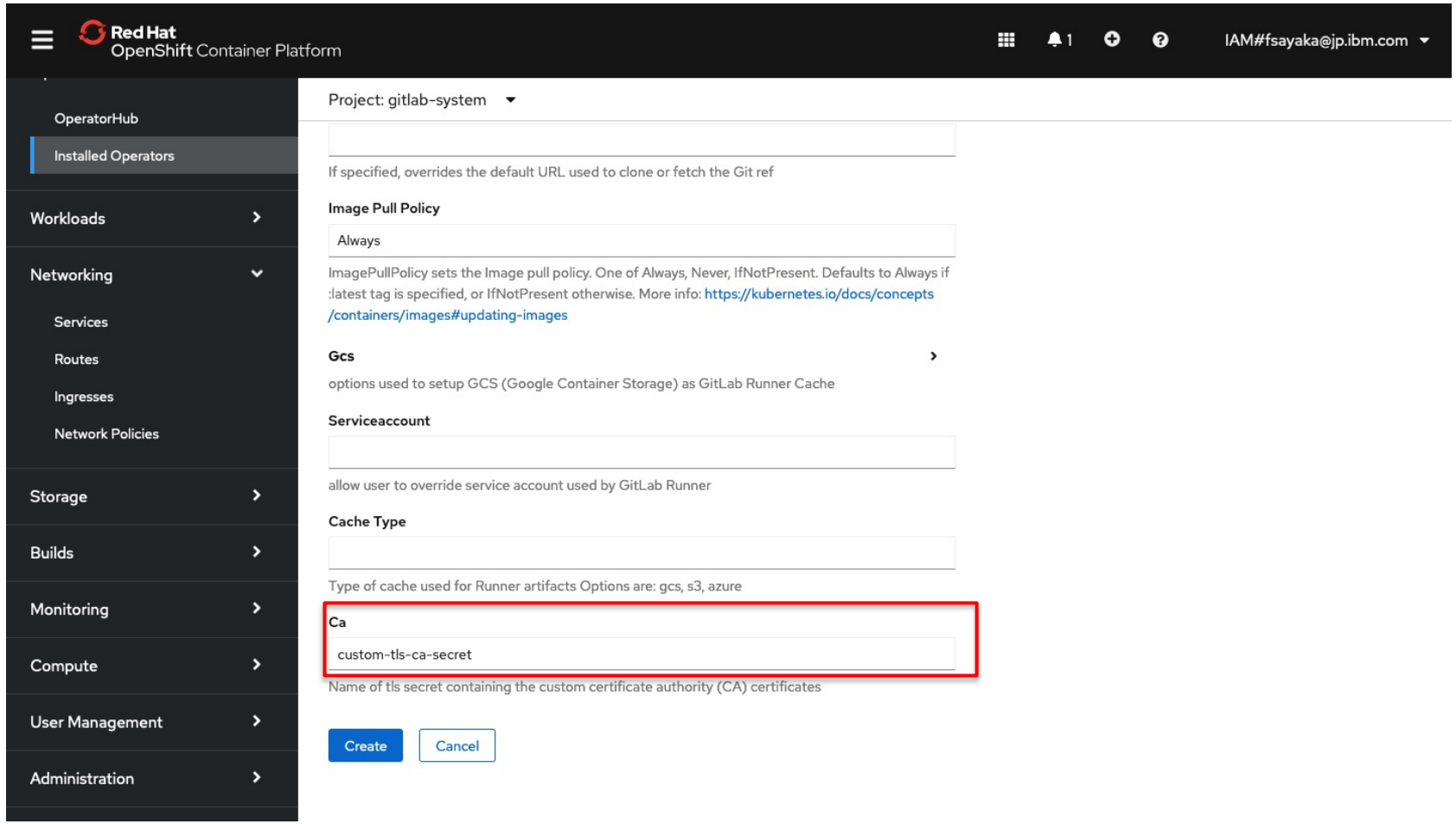
2.2 OperatorによるGitLab Runnerのインストール –4. GitLab Runnerの導入

6.表示された画面で以下の値を入力する
- 入力画面例



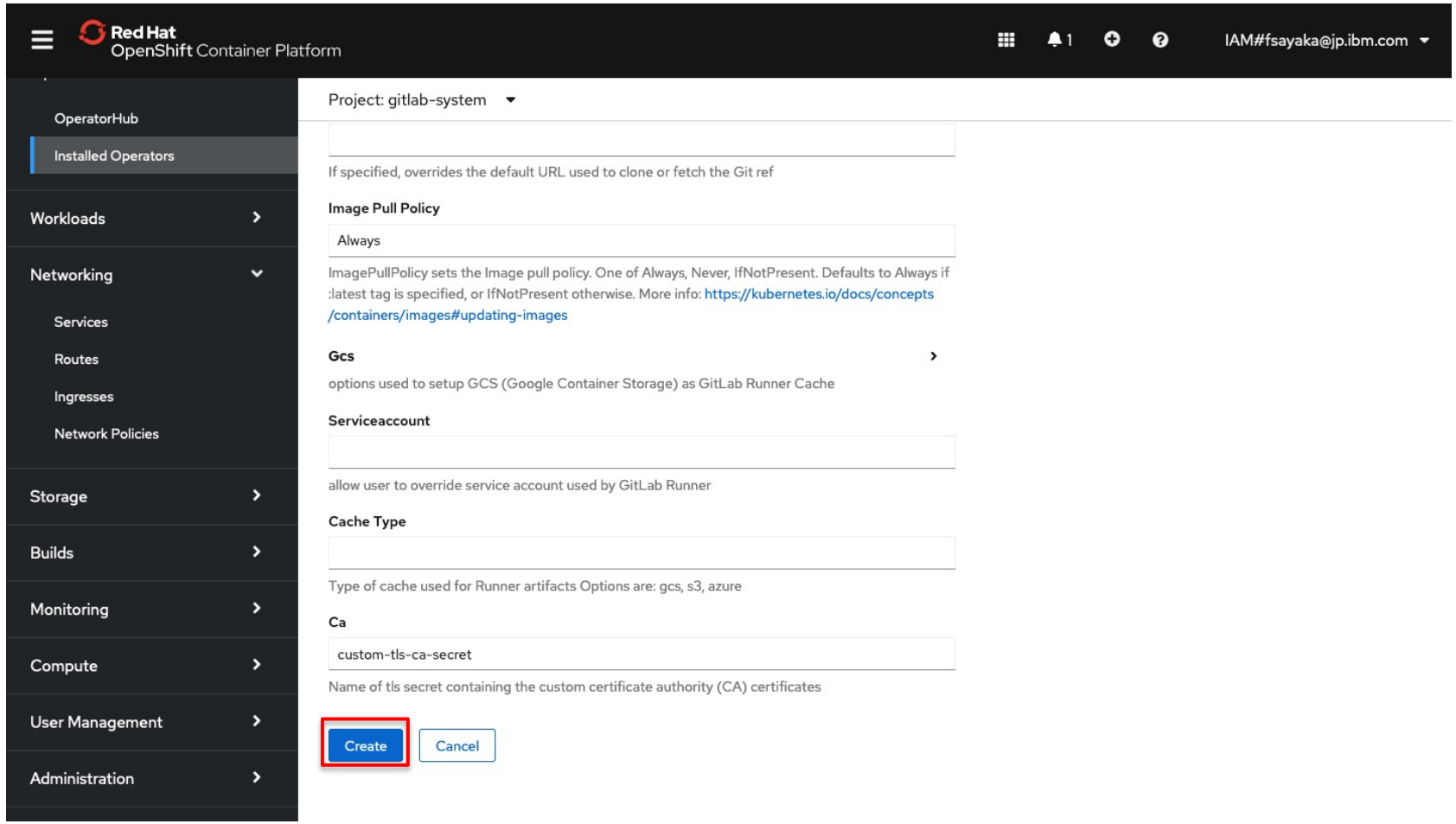
2.2 OperatorによるGitLab Runnerのインストール –4. GitLab Runnerの導入

6.表示された画面で以下の値を入力する
- 入力画面例



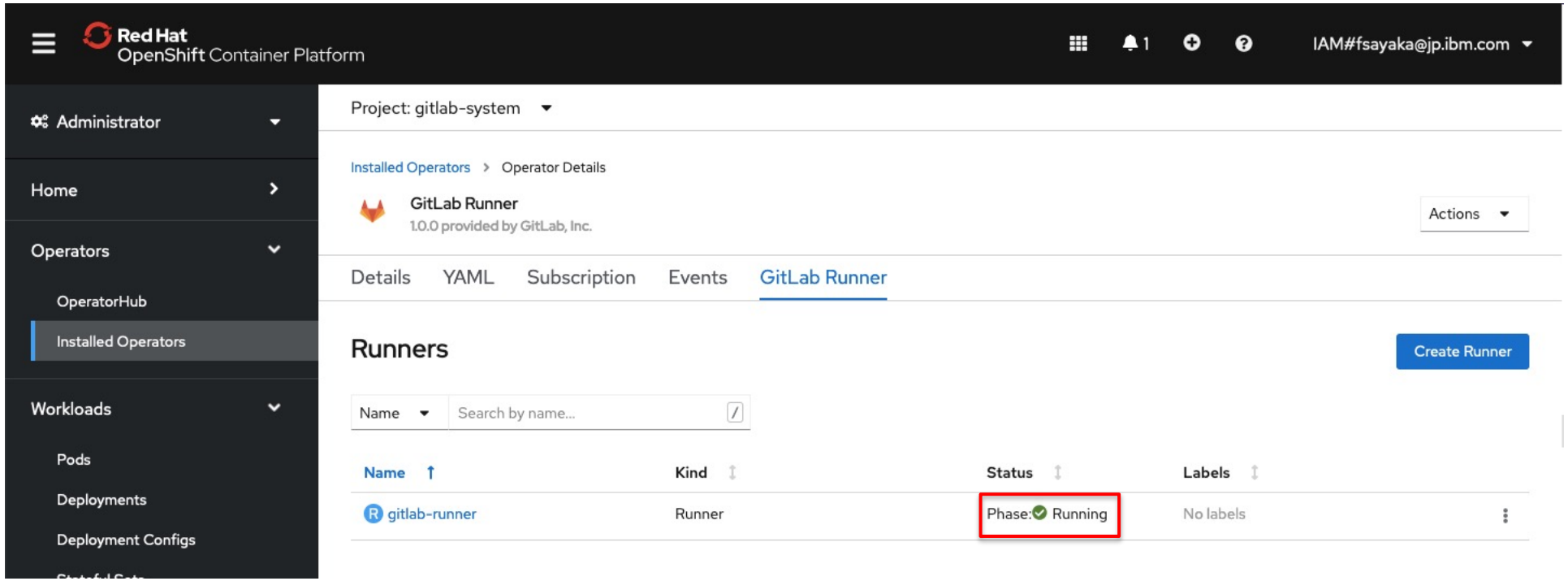
2.2 OperatorによるGitLab Runnerのインストール –4. GitLab Runnerの導入

7.入力が終わったら、「Create」をクリックする



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



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

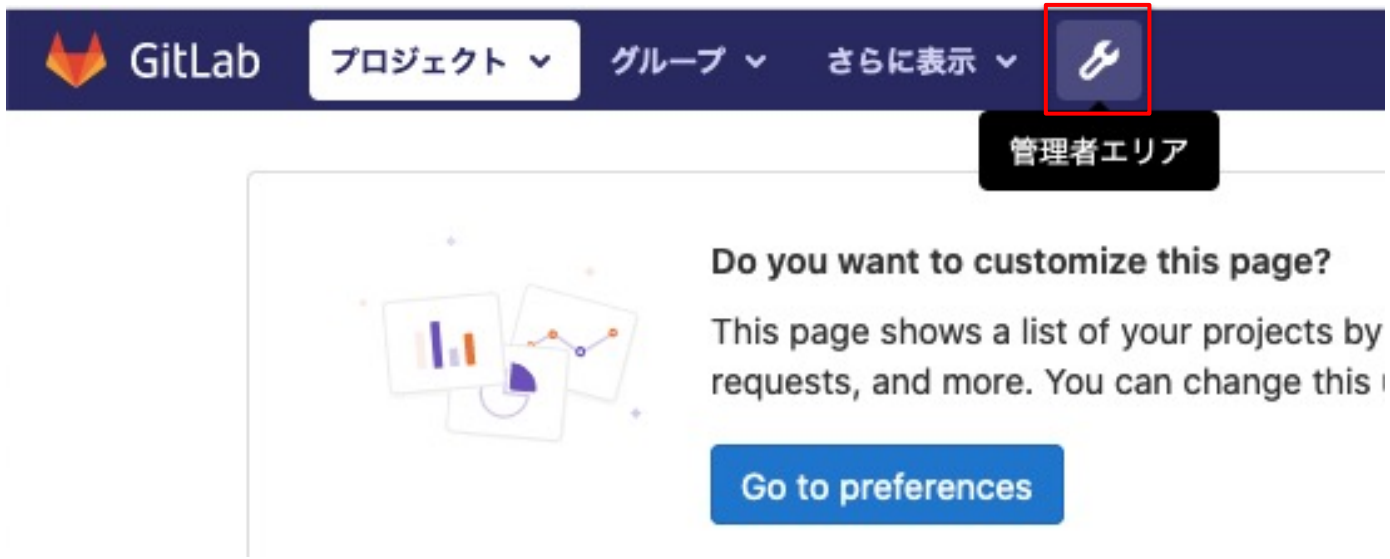
導入の流れは以下の通り

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

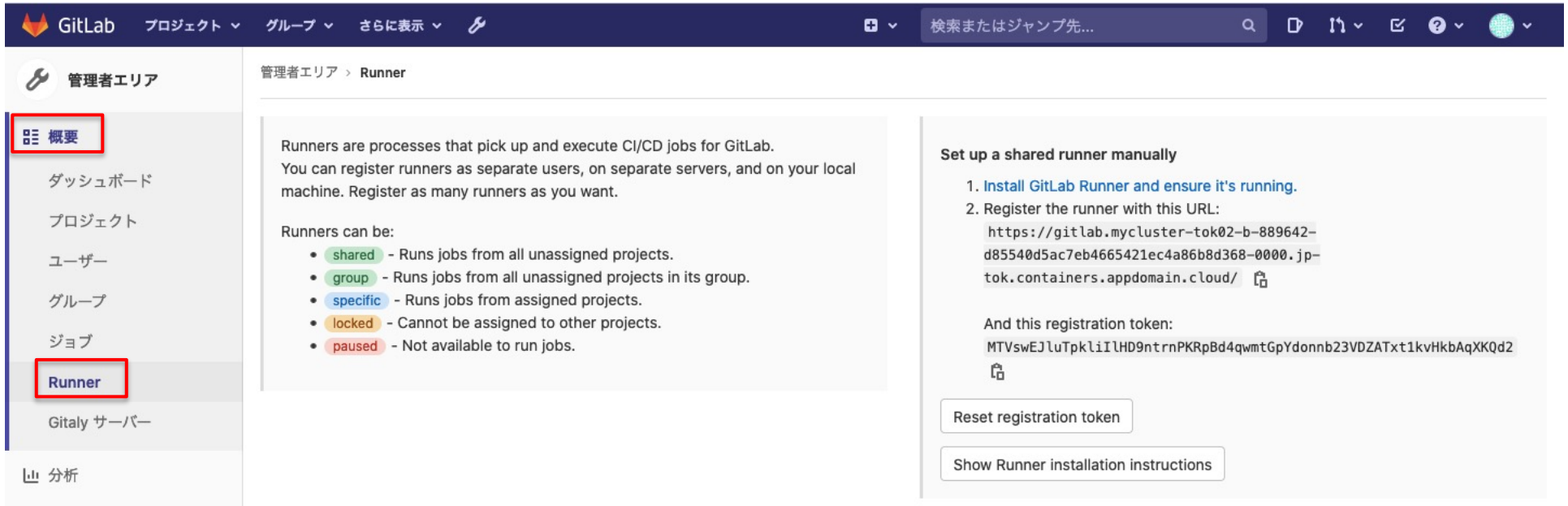
GitLabの画面から、GitLab Runnerが接続したことを確認する
以降は、共有Runnerとして登録した場合を例に説明する

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



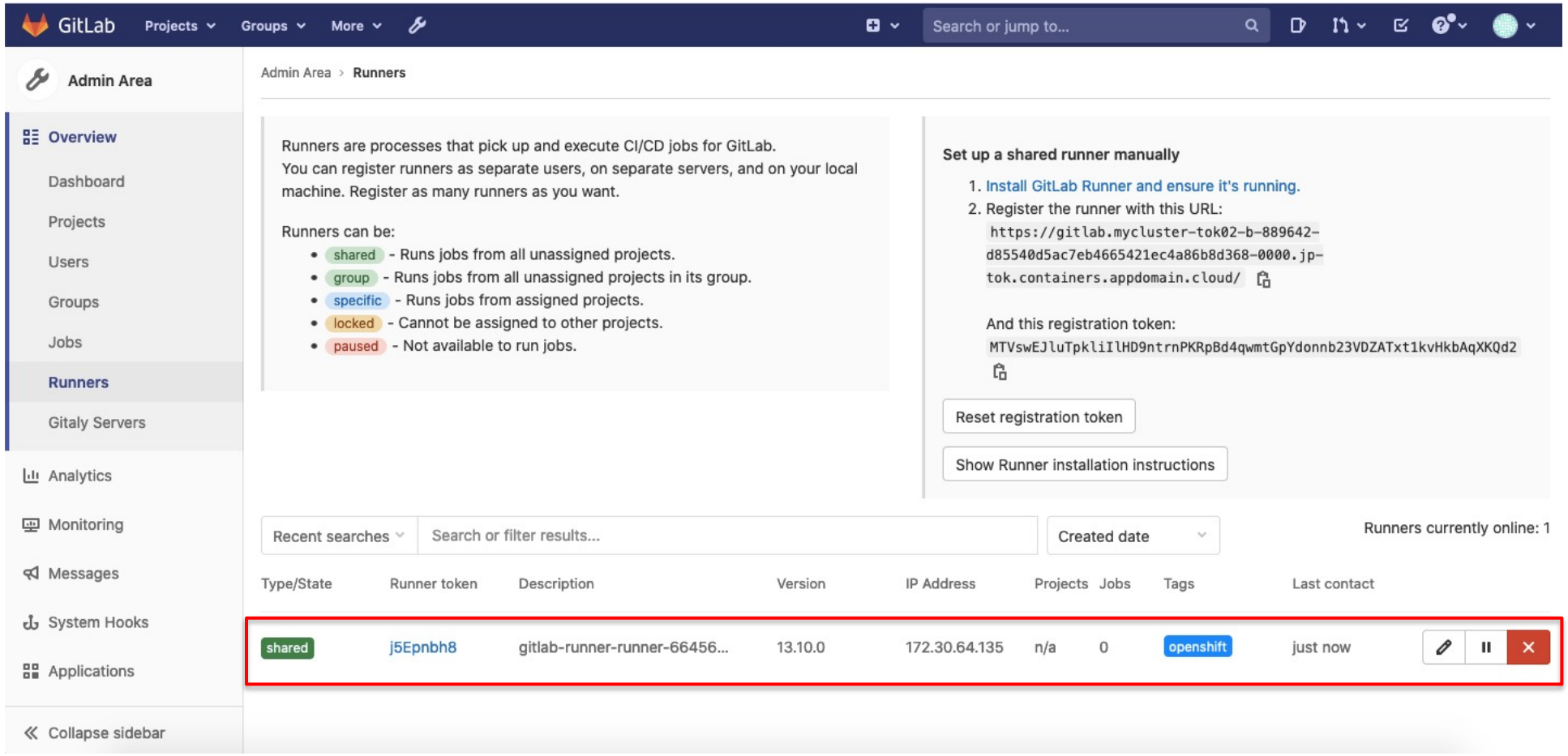
2.2 OperatorによるGitLab Runnerのインストール –5. 接続確認

2.「概要」の下の 「Runner」をクリックする



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

3.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



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

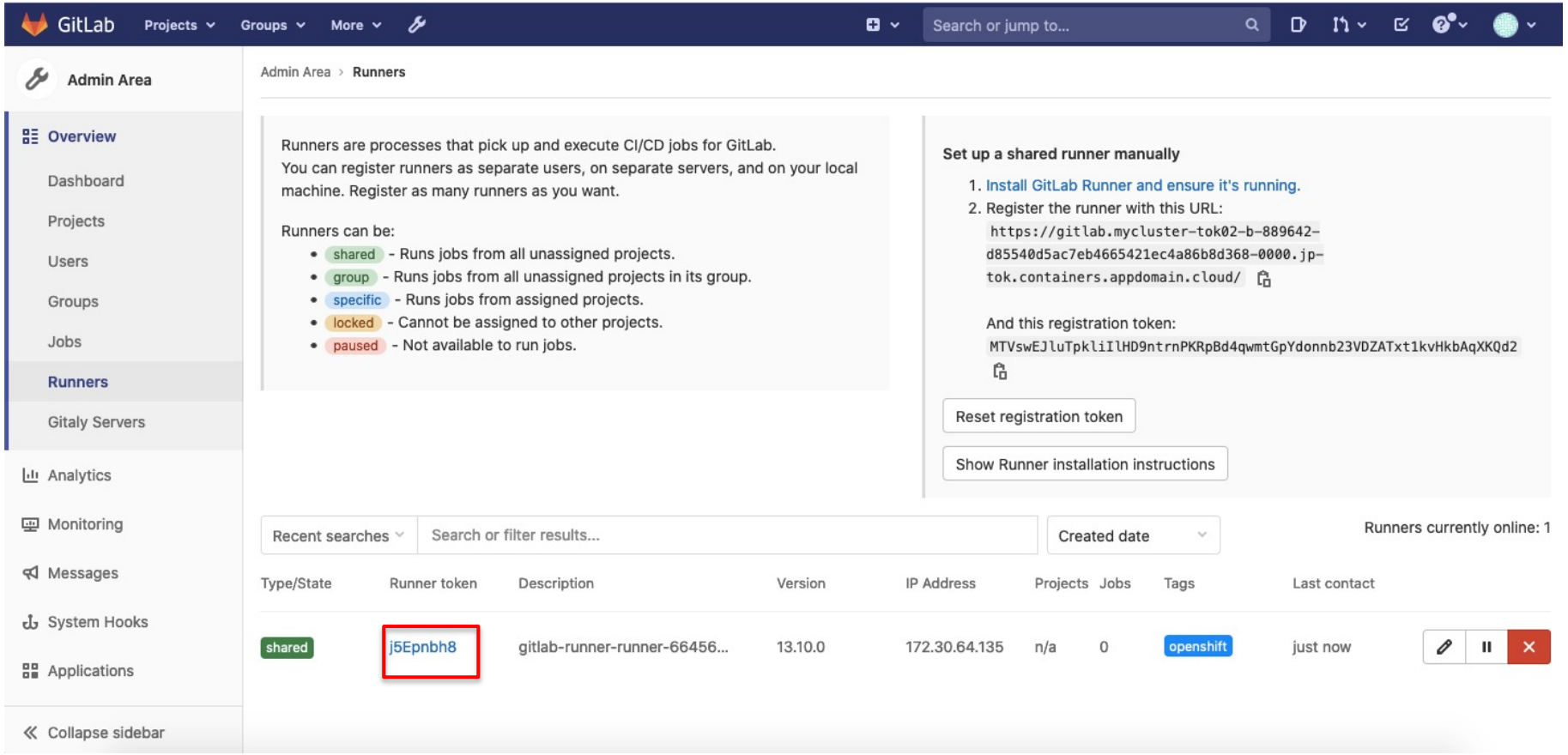
1. GitLab Runner Operatorの導入
2. GitLabRunner登録トークン、URLの確認とsecretの作成
3. （自己署名証明書を使用する場合）route証明書の確認とsecretの作成
4. GitLab Runnerの導入
5. 接続確認
6. **補足設定**

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

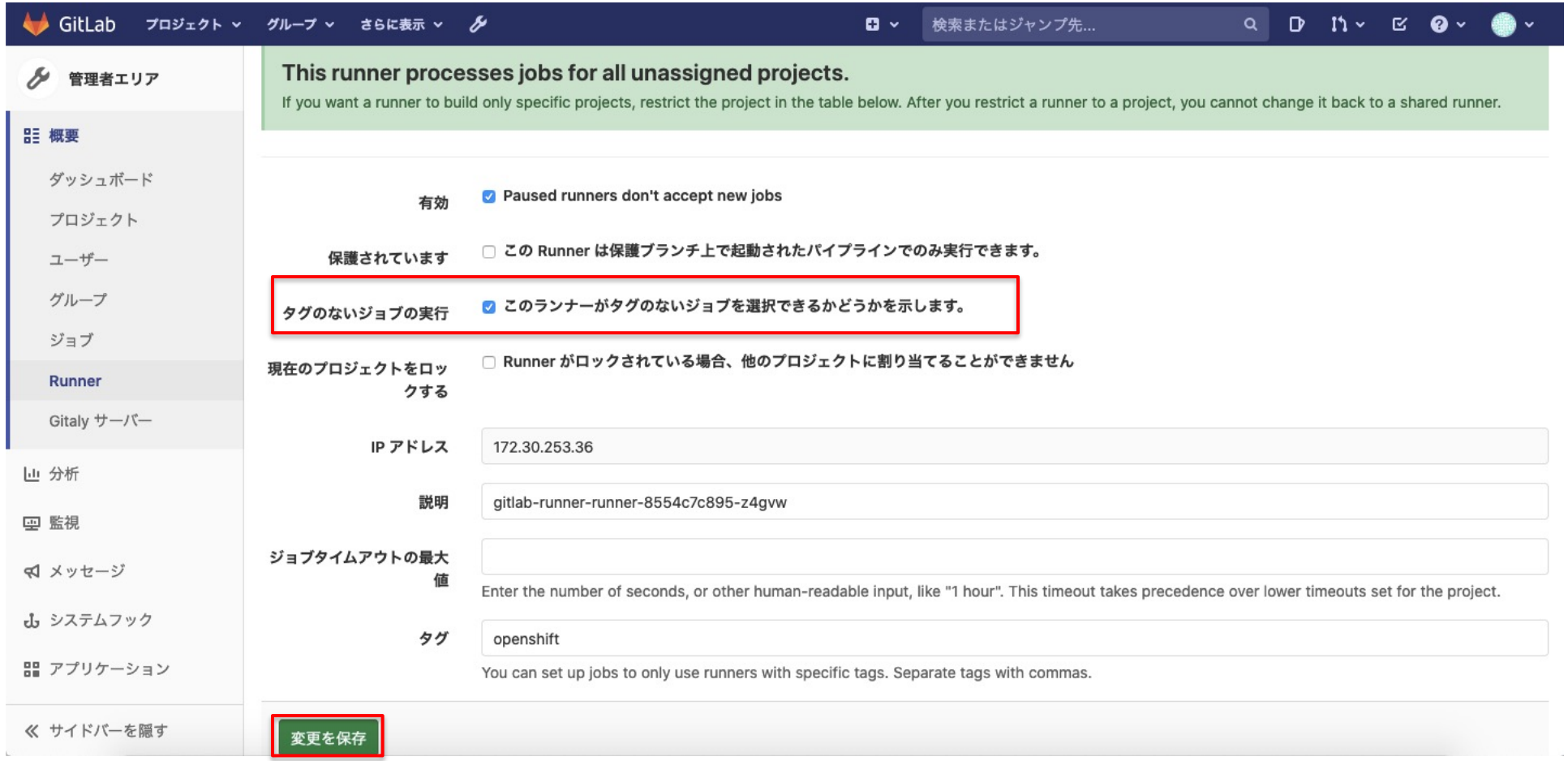
1.設定対象のRunnerのトークンのリンクをクリックする



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



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

- ライセンスの取得については別途営業担当者に確認のこと
- 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

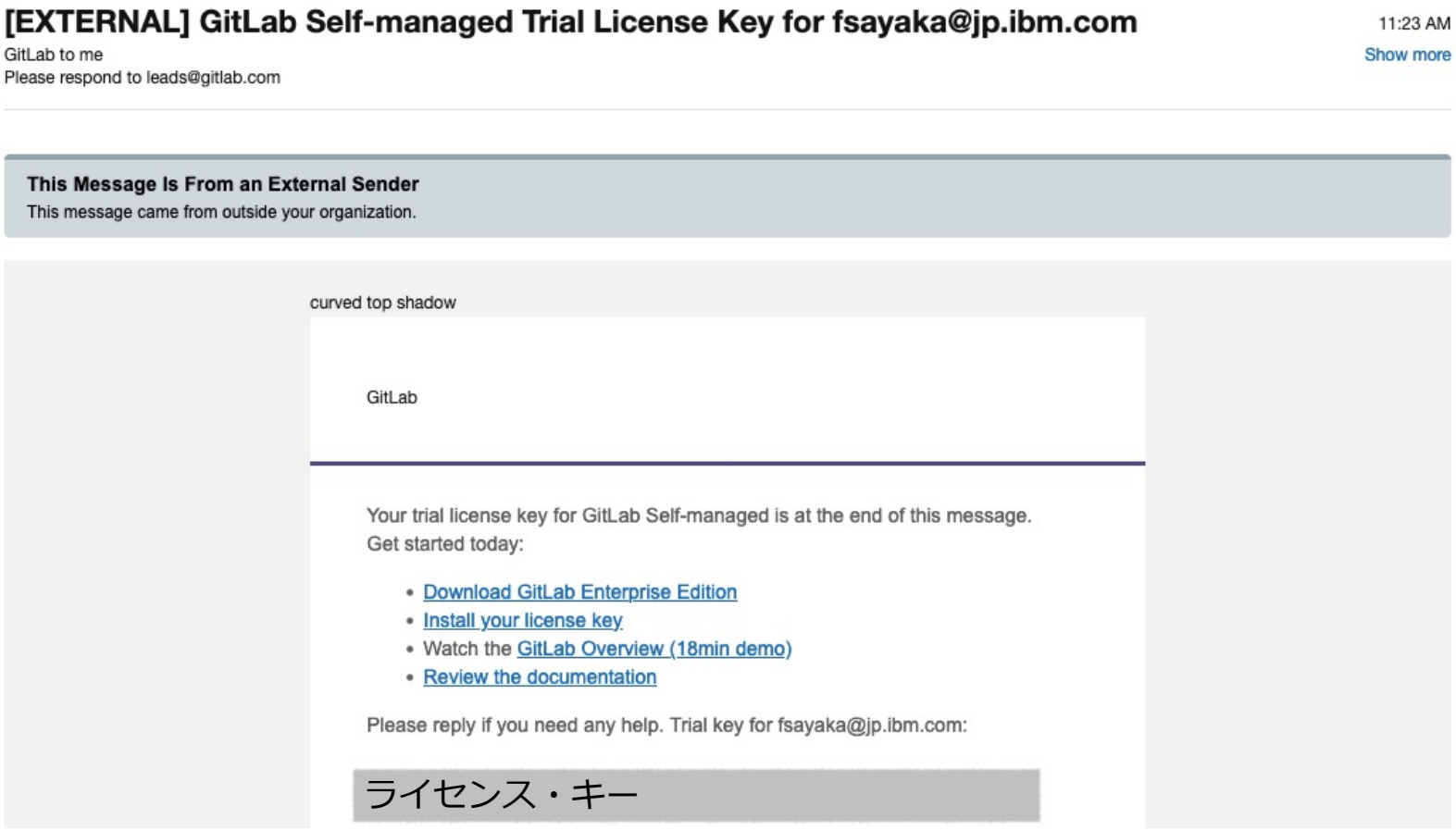
導入の流れは以下の通り

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

11:23 AM

GitLab to me

Show more

Please respond to leads@gitlab.com

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

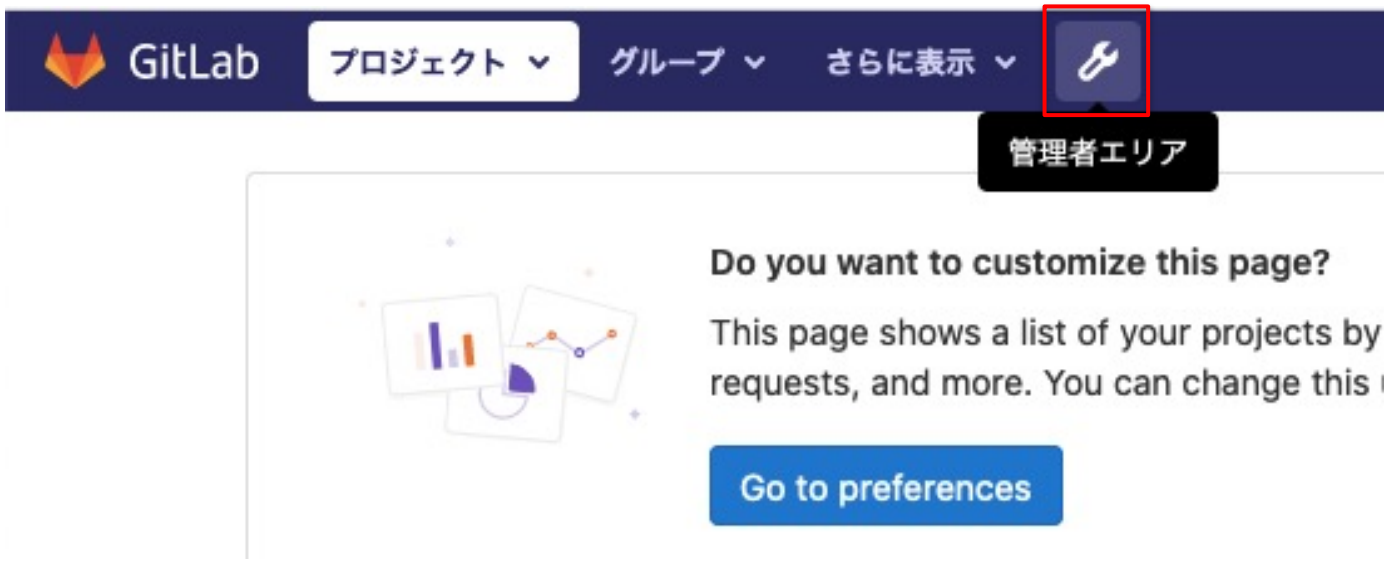
2.3 ライセンスの設定 -2.ライセンスの設定

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**

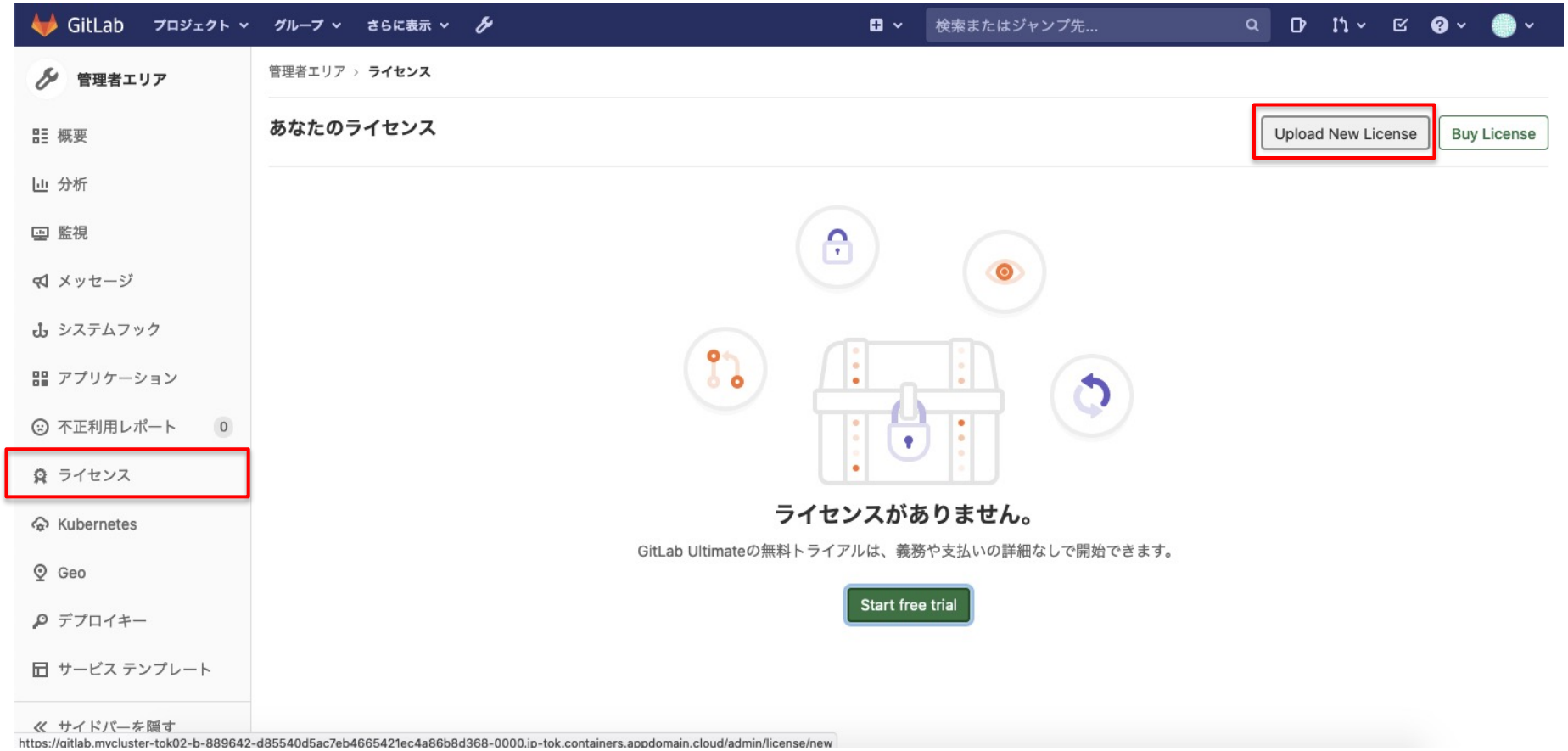
2.3 ライセンスの設定 -2.ライセンスの設定

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

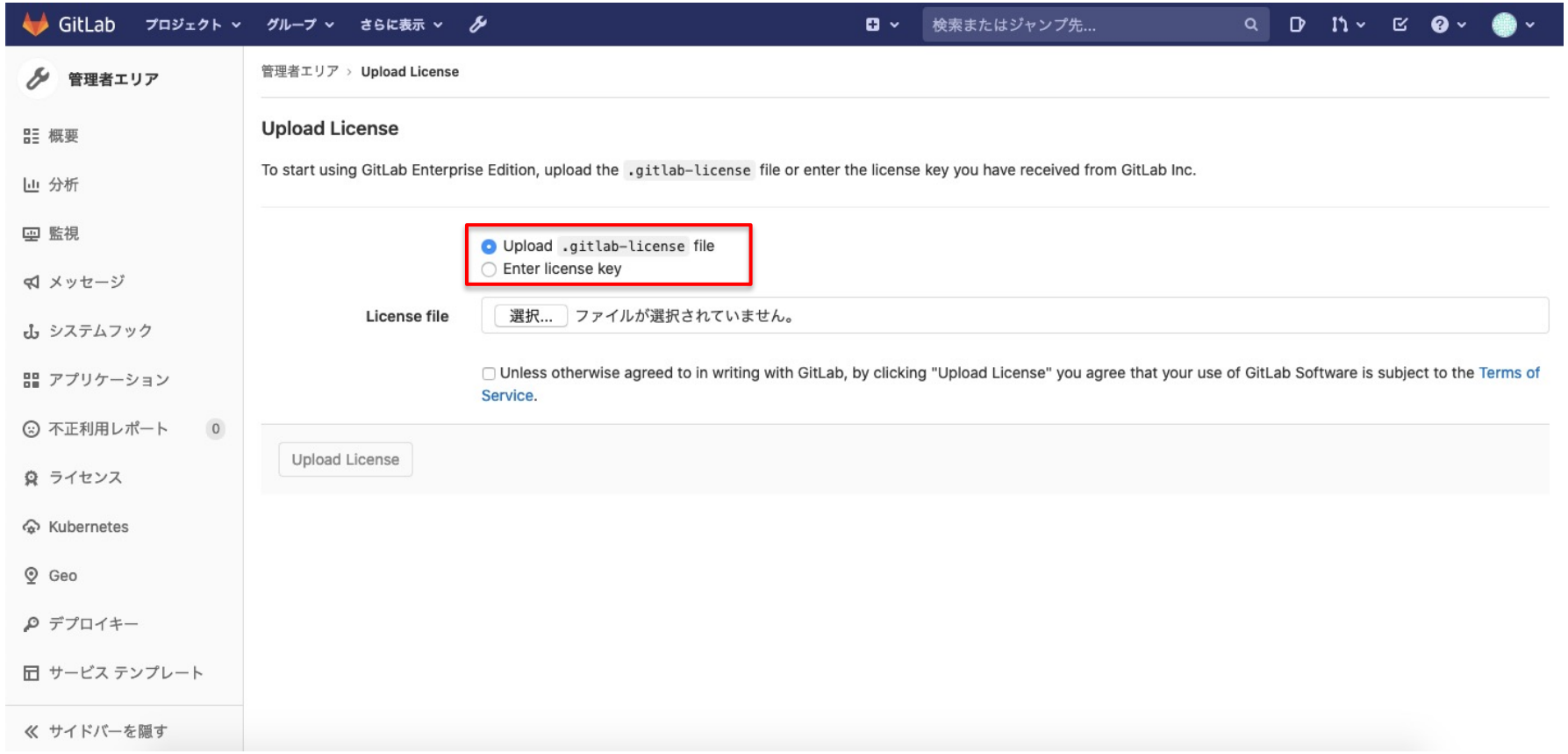


2.3 ライセンスの設定 -2.ライセンスの設定

3.「Upload License」画面が表示される

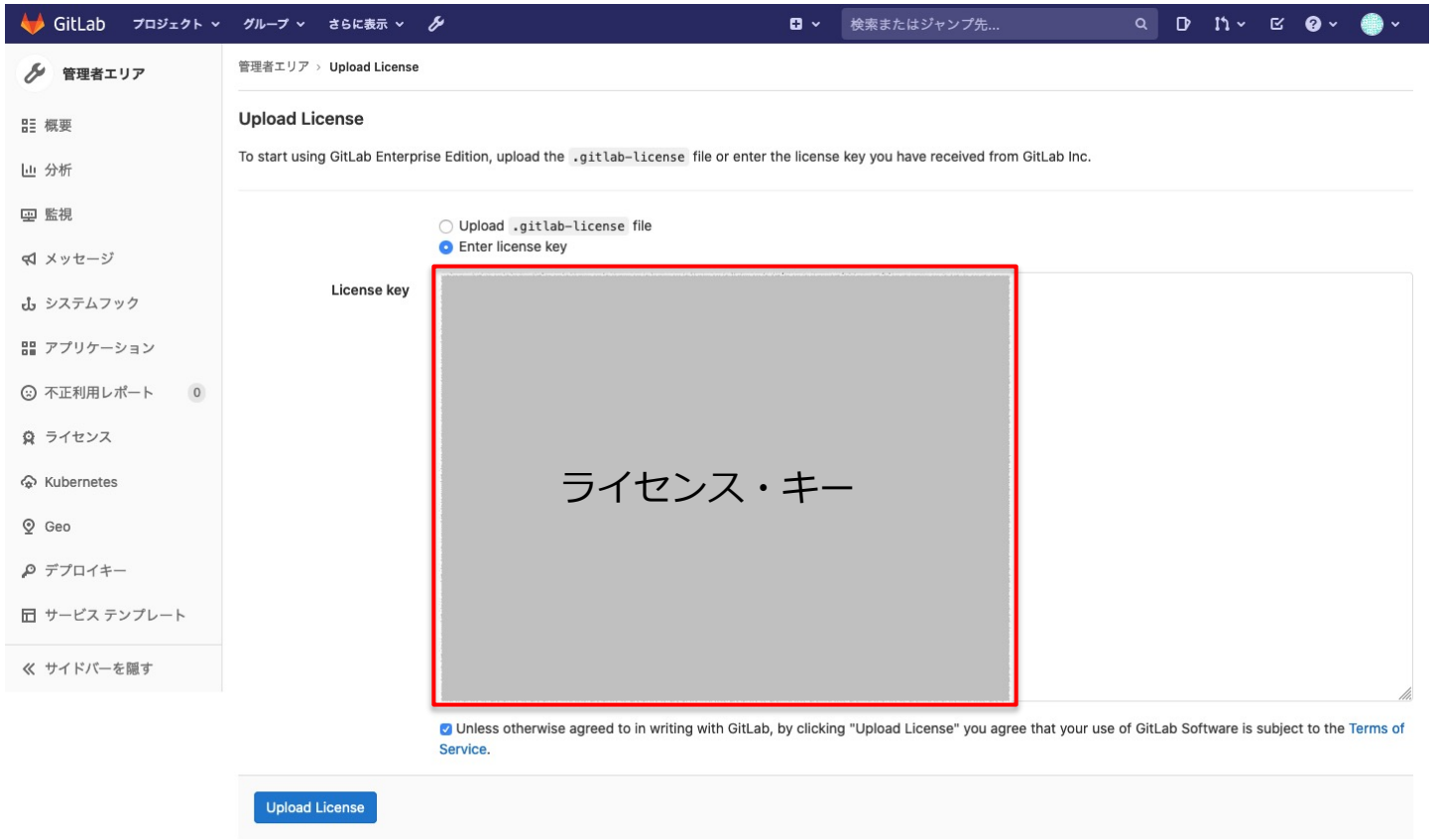
ライセンスの提供形態に合わせて以下のいずれかを選択する

- .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



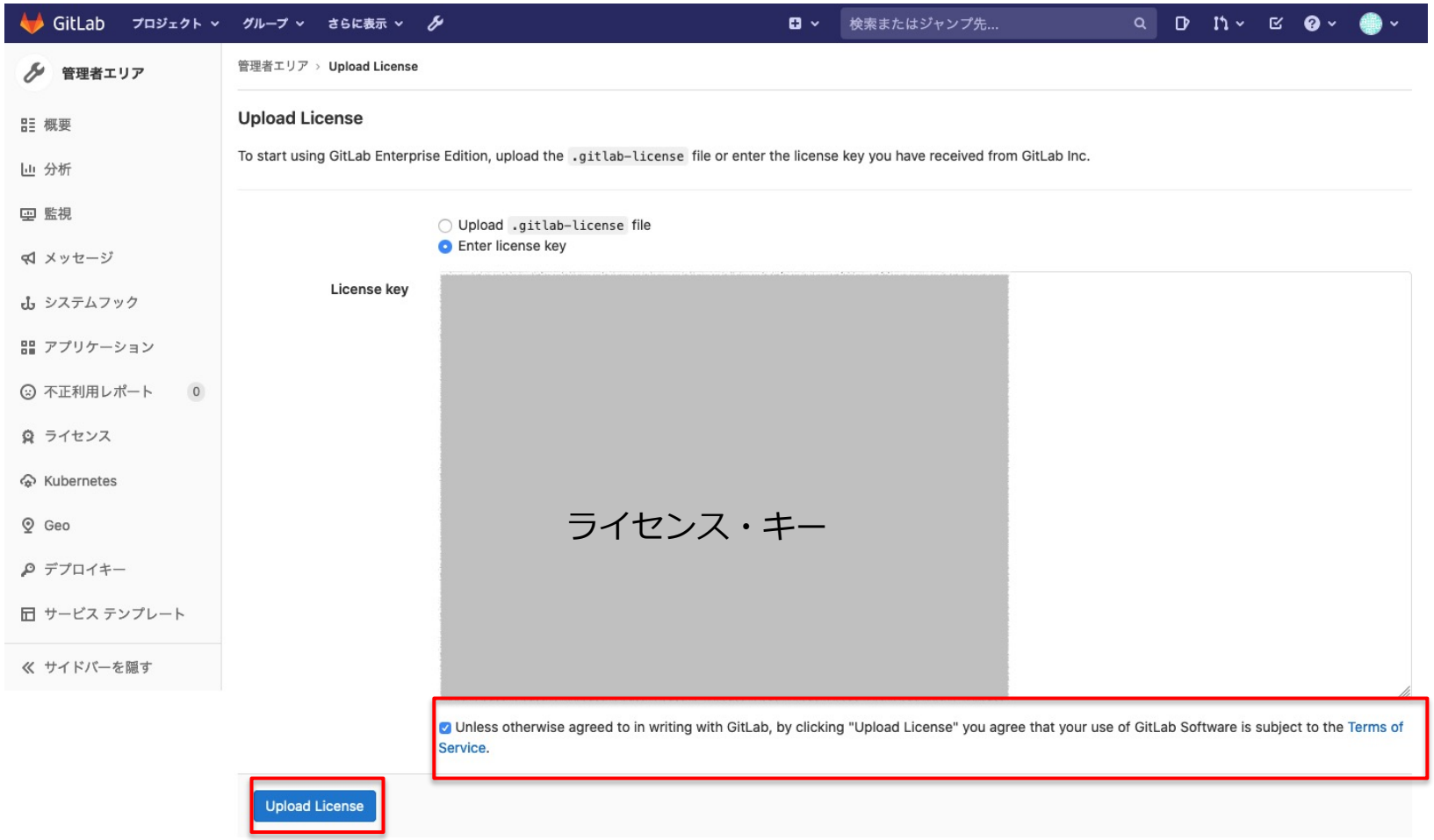
2.3 ライセンスの設定 -2.ライセンスの設定

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



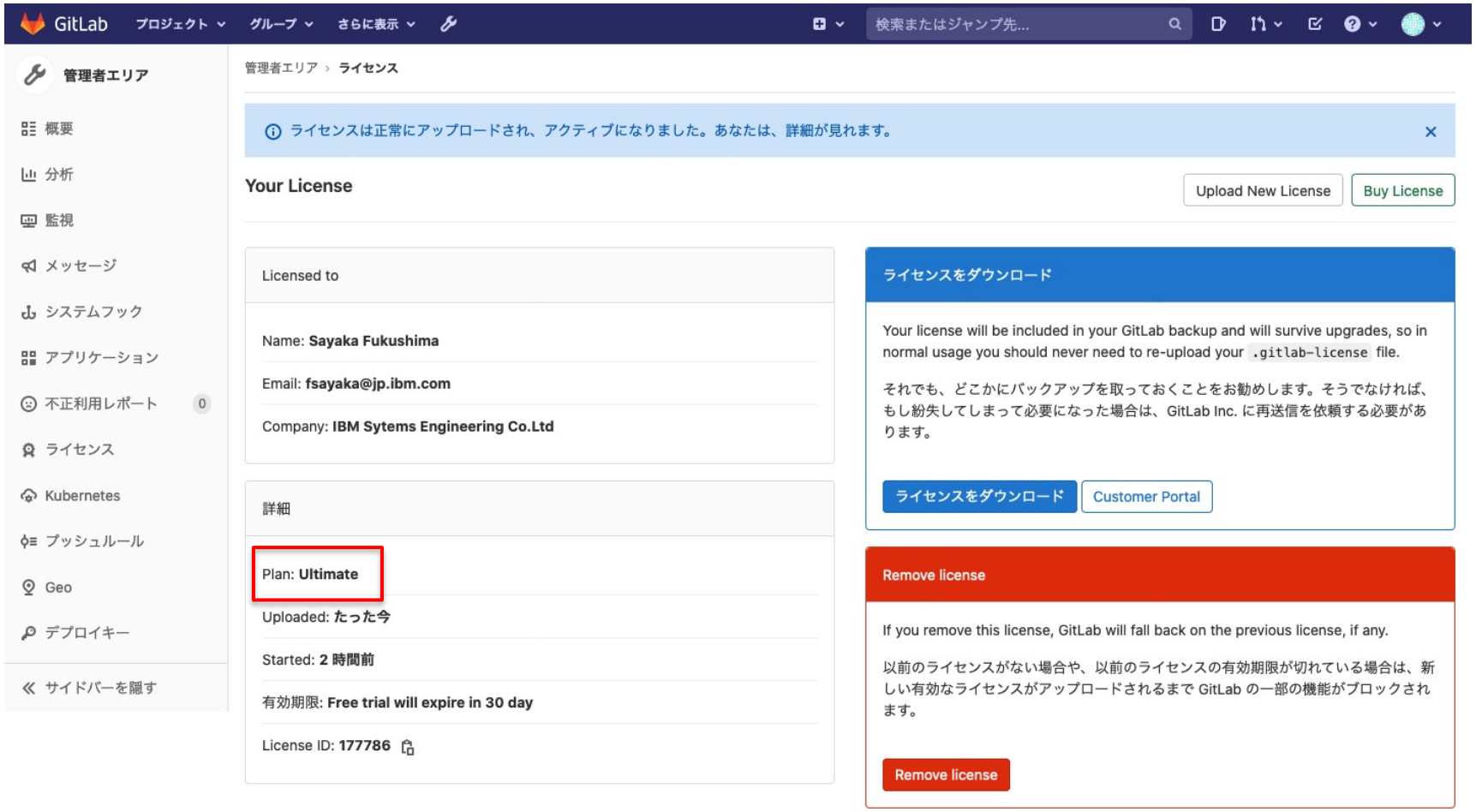
2.3 ライセンスの設定 -2.ライセンスの設定

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



2.3 ライセンスの設定 -2.ライセンスの設定

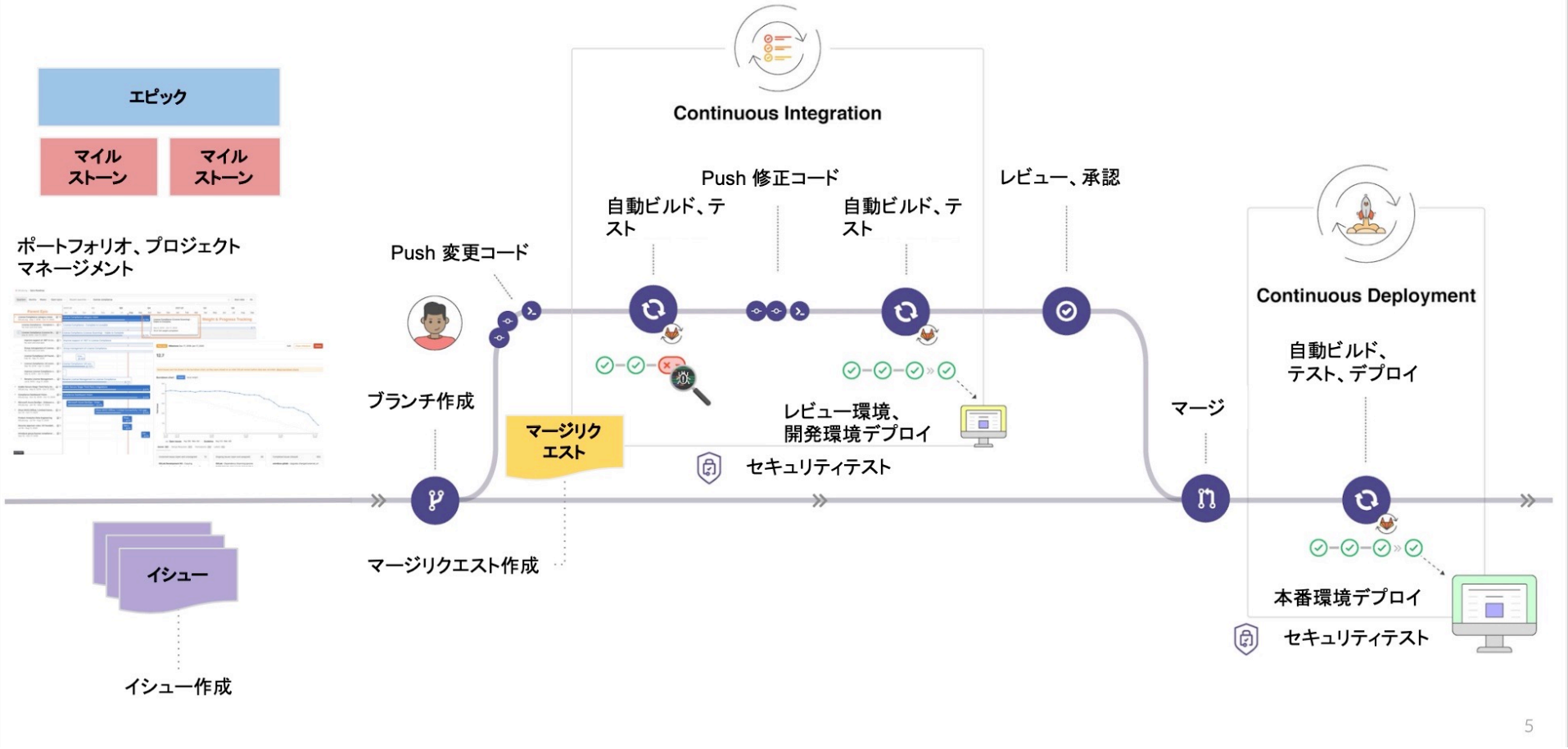
6. ライセンスのアップロードが完了するとライセンス情報が表示される
Planに「Ultimate」と表示されればUltimateのライセンスが設定されている



- 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

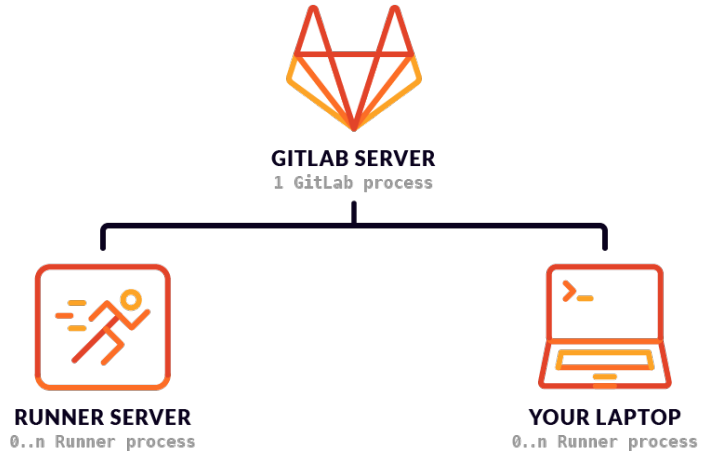


GitLab合同会社「GitLabのマージリクエストとCI/CDを活用した並行開発」より抜粋

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。

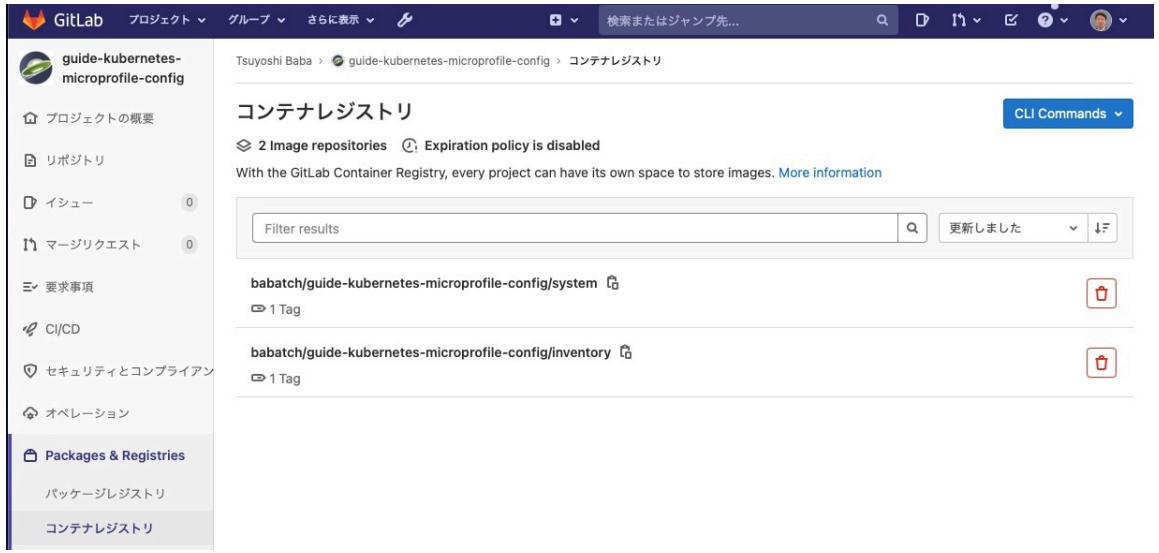
- GitLab Runner は…
  - ビルドを実行するアプリケーション。
  - GitLab とは別にインストールする必要がある。
  - インストール後、GitLab サーバーに登録すると使用可能になる。



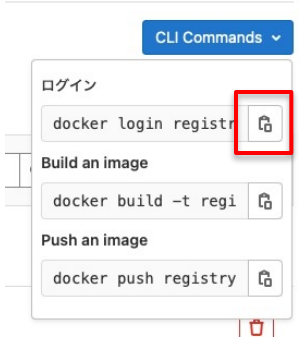
https://www.gitlab.jp/stages-devops-lifecycle/continuous-integration/

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>

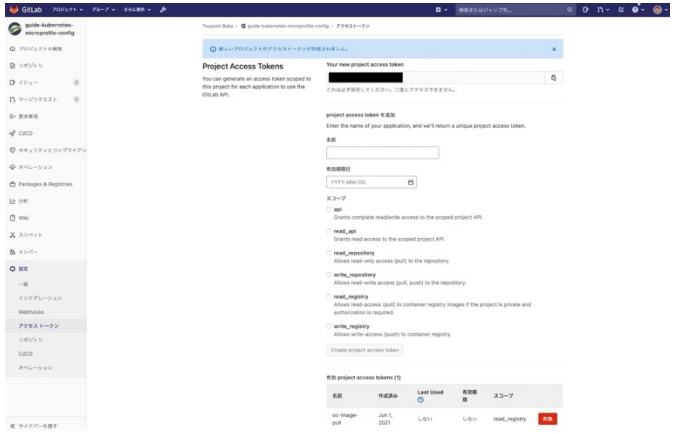


GitLab コンテナレジストリのイメージプルシークレット

- 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
    - 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)
```



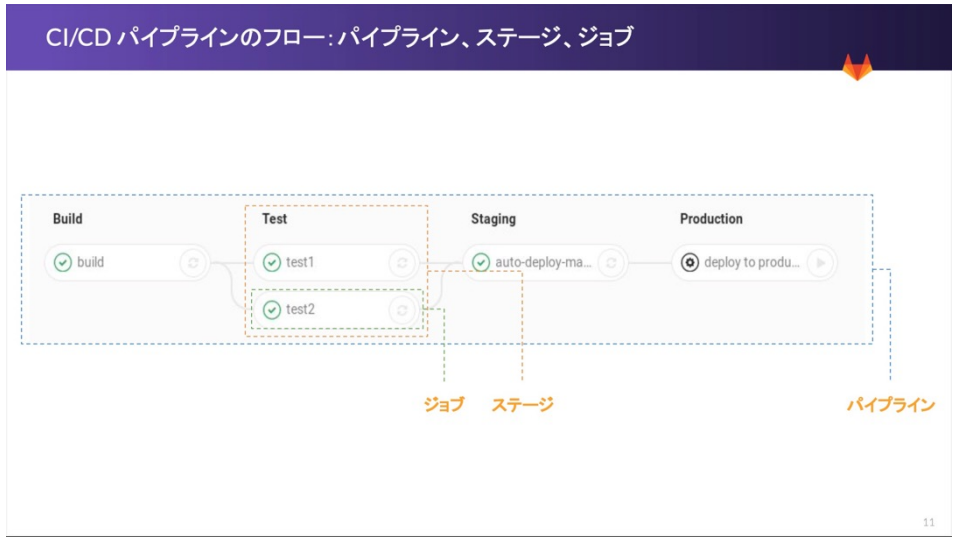
Project access token の生成

- 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

.gitlab-ci.yml

```
stages:
  - build
  - test
  - staging
  - production
```
ステージ一覧

```
build:
  stage: build
  …
```
属するステージ名 / ジョブ

```
test1:
  stage: test
  …
```
属するステージ名 / ジョブ

```
test2:
  stage: test
  …
```
属するステージ名 / ジョブ

```
auto-deploy-main:
  stage: staging
  …
```
属するステージ名 / ジョブ

```
"deploy to production":
  stage: production
  …
```
属するステージ名 / ジョブ

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 cache: キーワードを用いて定義する。	ジョブに artifacts: 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 (dependencies:) 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。

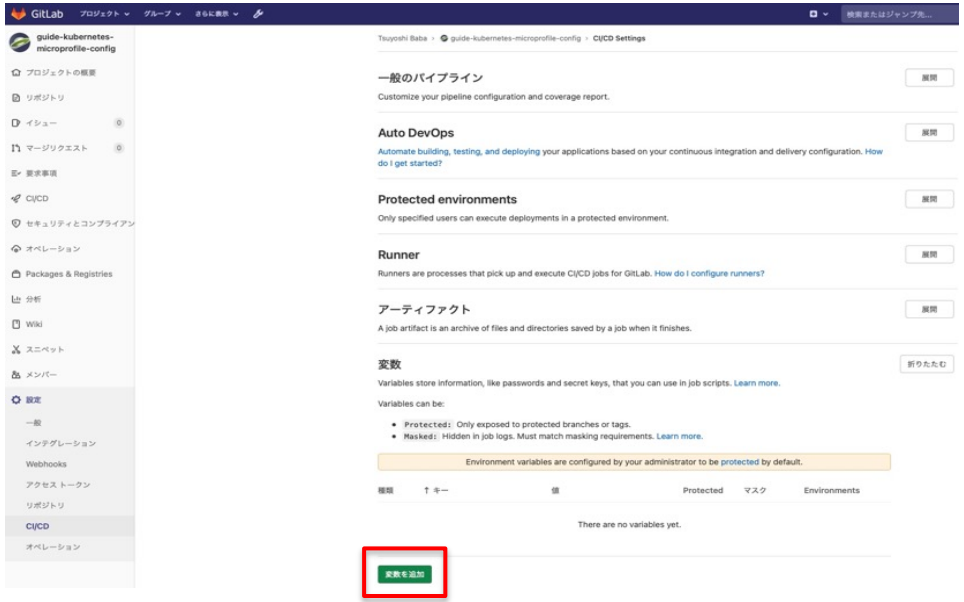
```
job_name:
  script:
    - export
```

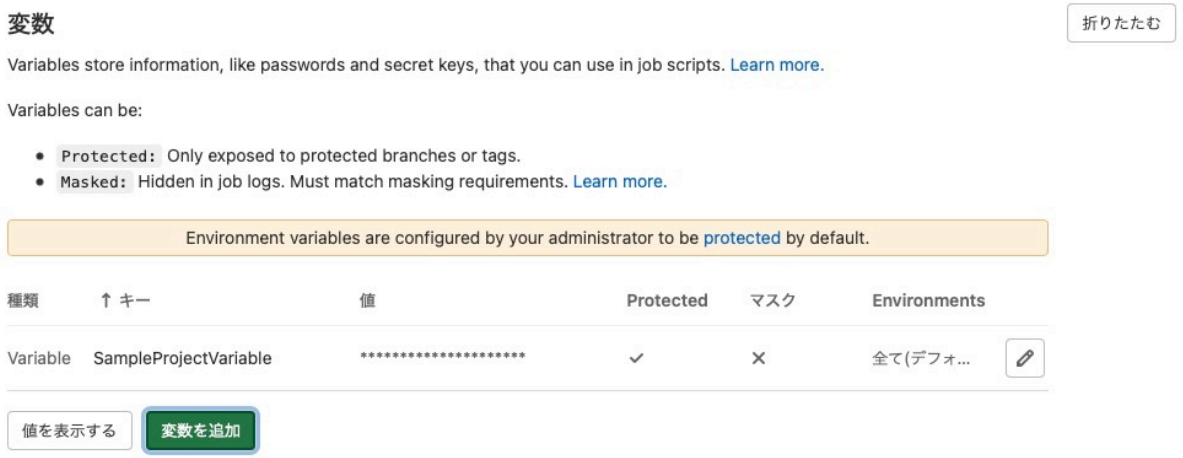
- 値を確認した例

```
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

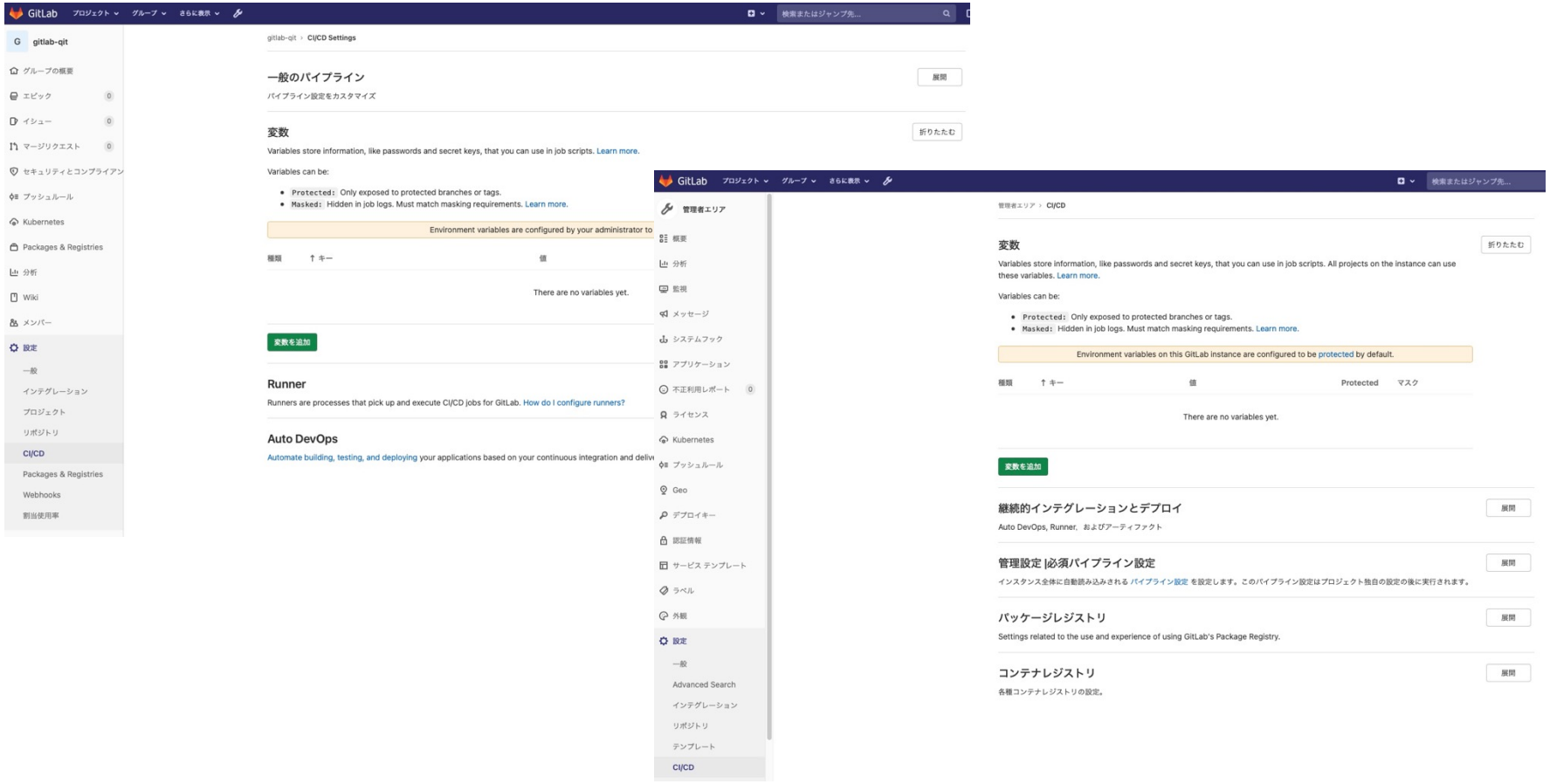
- プロジェクト内で使用可能な CI/CD 変数
- プロジェクト設定 → CI/CD → 変数 から設定することができる。





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



- 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リポジトリー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. リポジトリー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リポジトリーへのコミット
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. **リポジトリー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リポジトリーへのコミット
5. 結果の確認

3.3. GitLab CI/CD利用ステップ–1.リポジトリー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ー（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、新規リポジトリー「guide-kubernetes-intro」を作成し、以下のリポジトリー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.リポジトリー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リポジトリー作成ディレクトリ>
```

3. Git リポジトリーを作成するディレクトリーへ移動（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ーをcloneする

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

5. cloneしたリポジトリーのディレクトリーに移動する

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

3.3. GitLab CI/CD利用ステップ –1.リポジトリー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リポジトリーの登録名を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ー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.リポジトリー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ーへコードをアップロード（git push）する

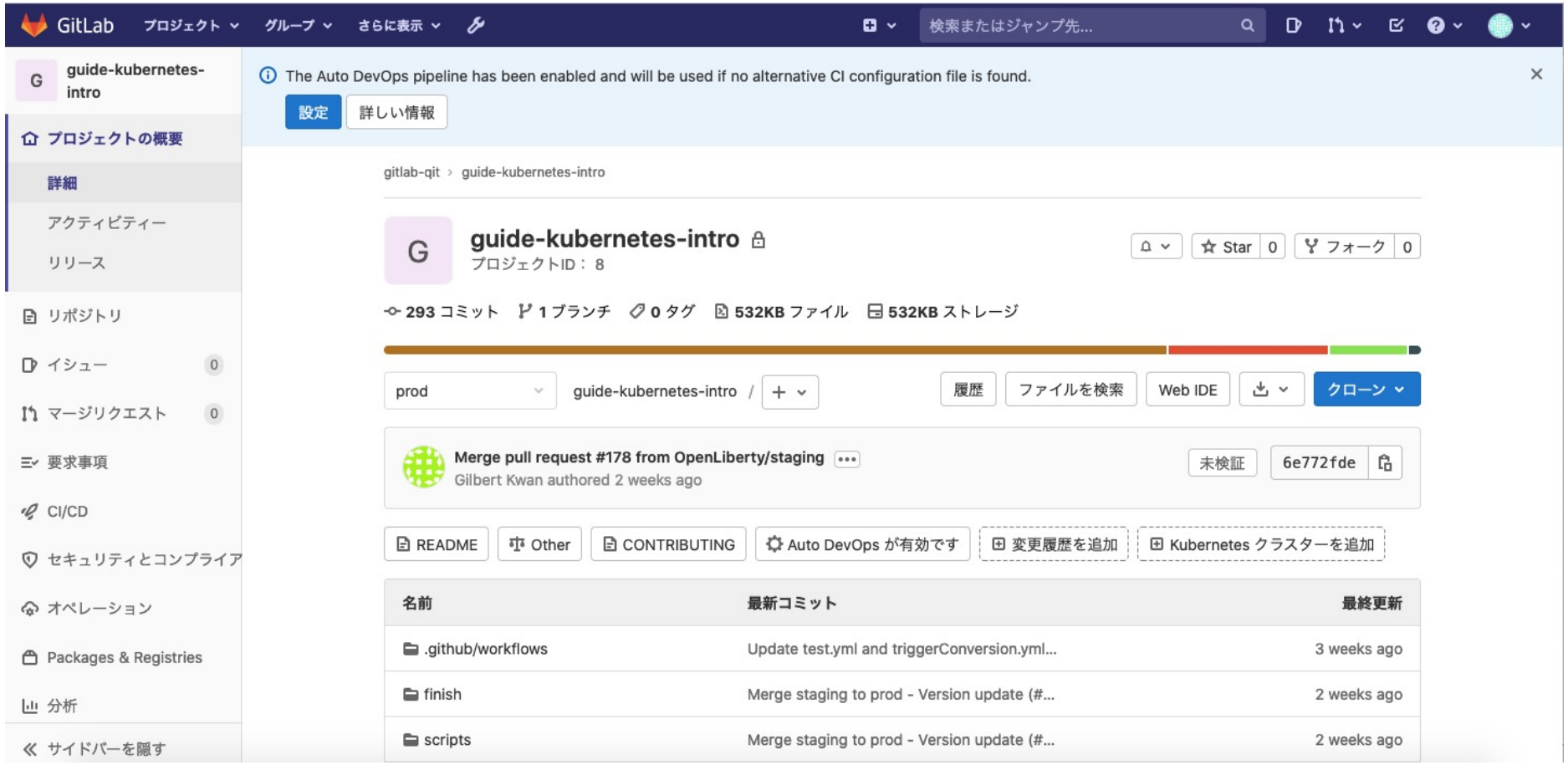
```
git push -u origin --all
```

– 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.リポジトリー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ーにソースコードが格納されたことを確認する



3.3. GitLab CI/CD利用ステップ–1.リポジトリー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. リポジトリー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リポジトリーへのコミット
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

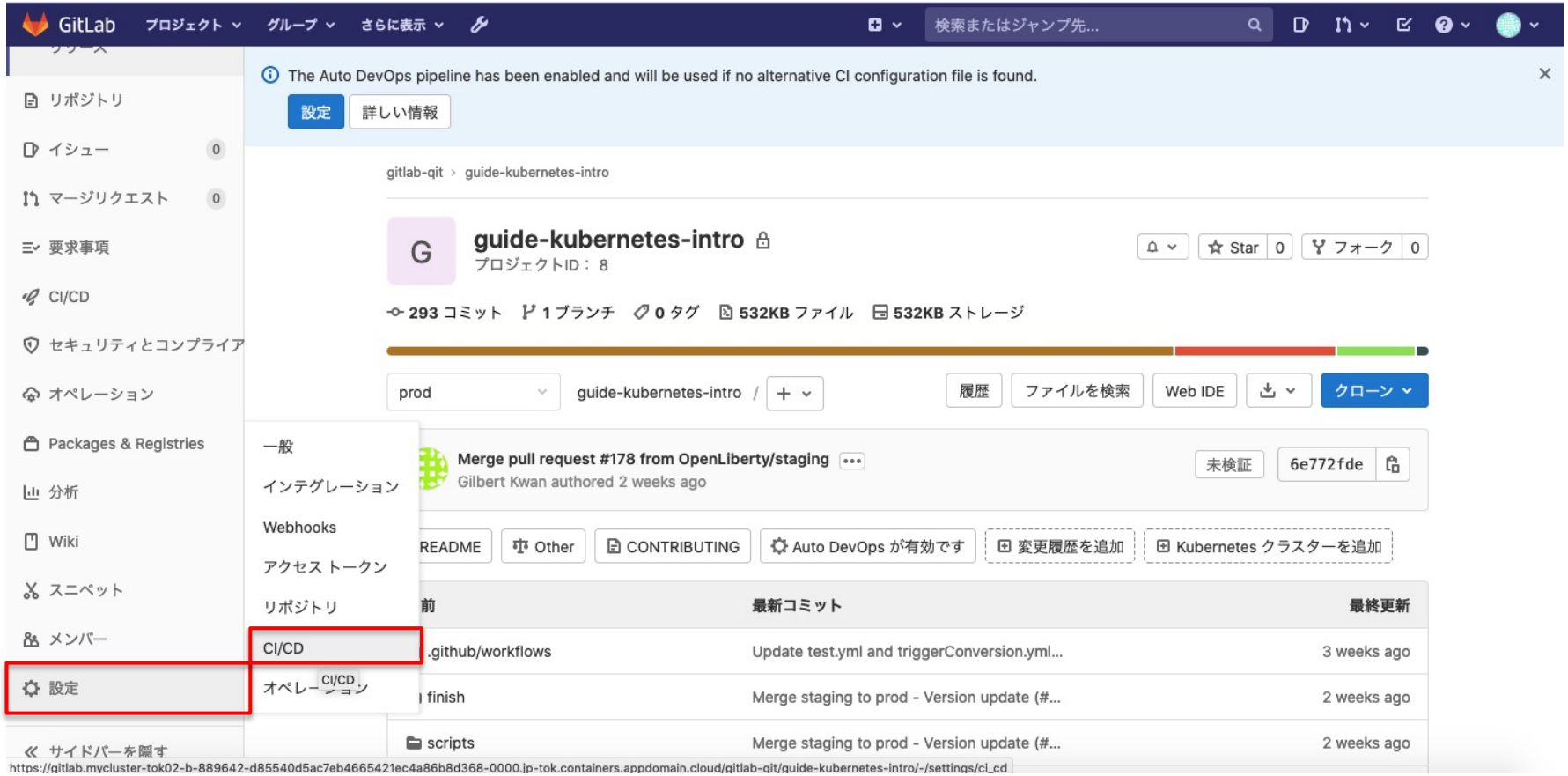
複数のパイプラインやジョブで利用するデータをプロジェクト変数として登録する。
また、APIキーなどの秘密情報をマスキングする場合にも変数を利用する

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

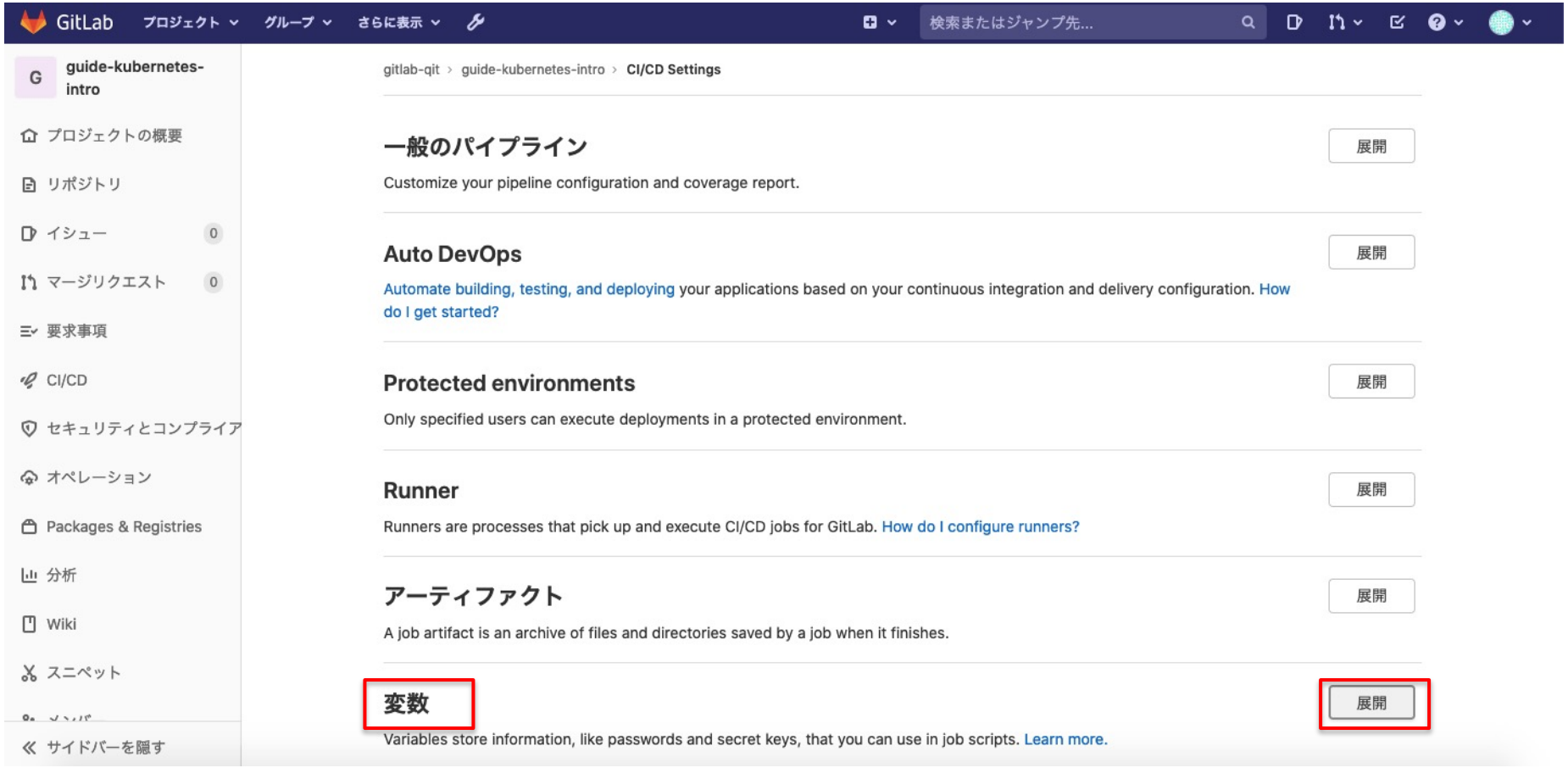
3.3. GitLab CI/CD利用ステップ –2.プロジェクト変数の追加

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



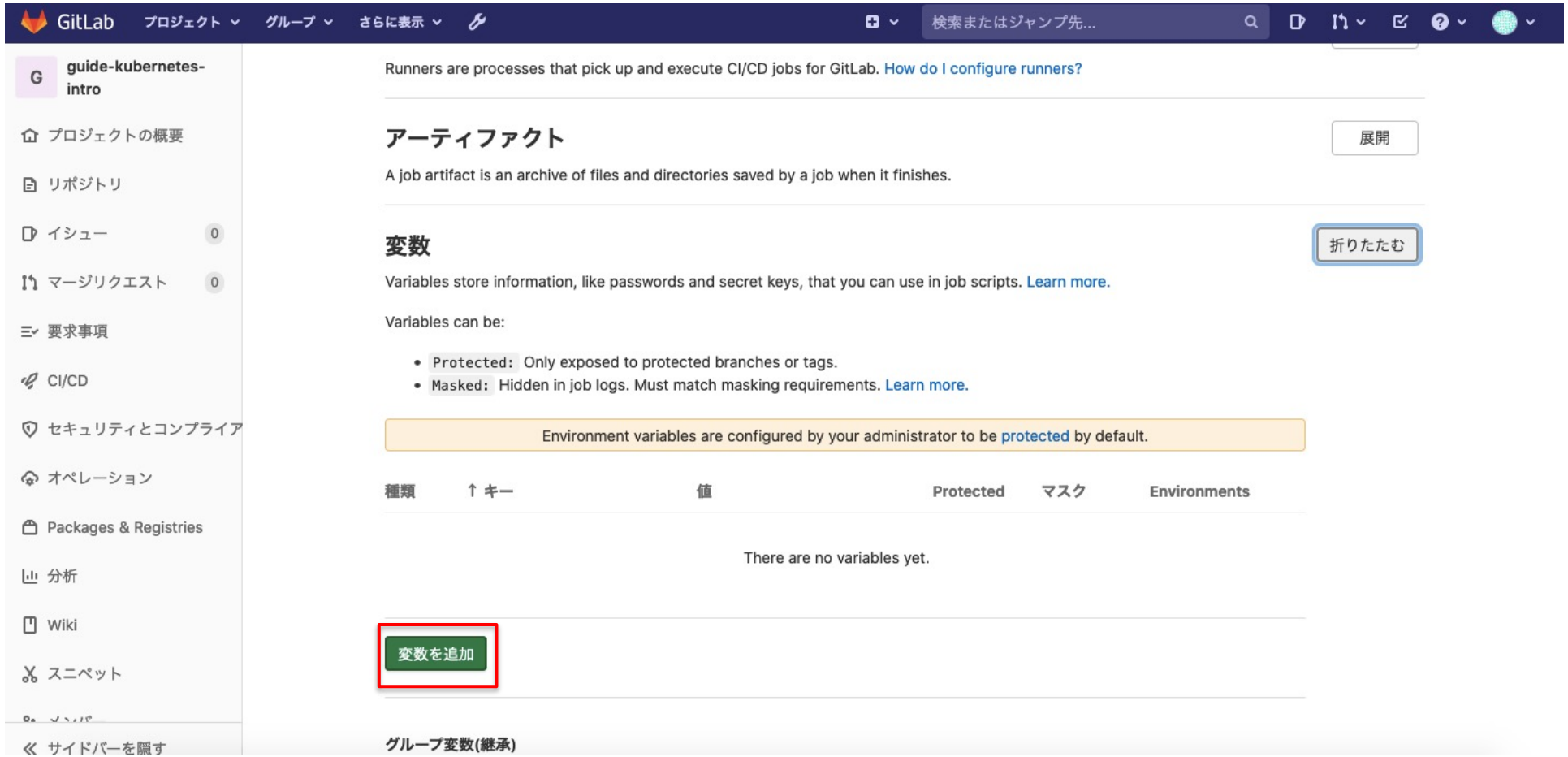
3.3. GitLab CI/CD利用ステップ –2.プロジェクト変数の追加

4. 表示された画面で変数の「展開」をクリックする



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



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

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

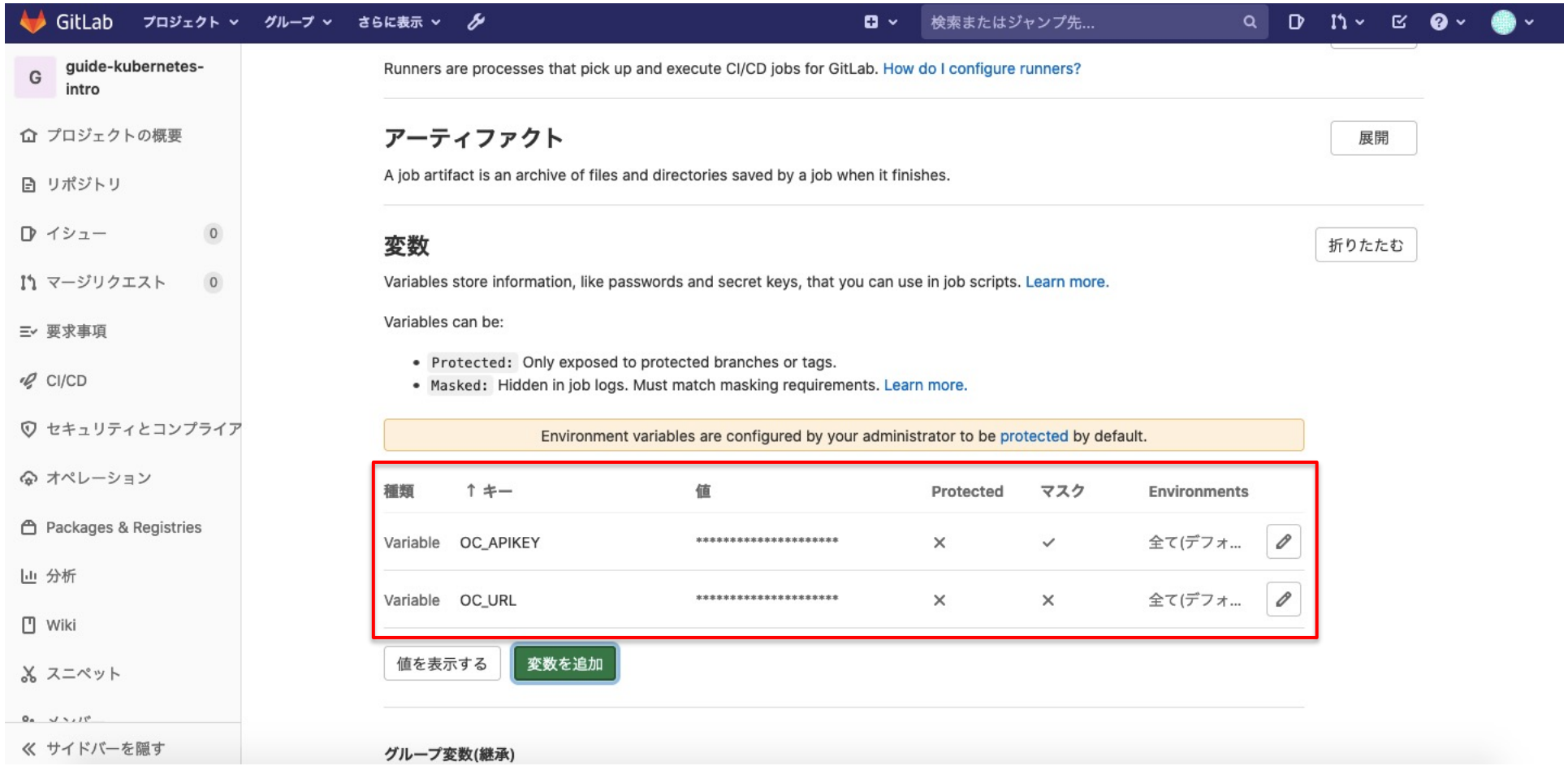
6. 変数のキーや値を設定する
   – 例、oc loginコマンドでのAPIキーを格納する変数
     秘密情報のため「Mask variables」にチェック

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

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



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. リポジトリー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リポジトリーへのコミット
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ではリポジトリー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

リポジトリー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リポジトリーのコードに対して.gitlab-ci.yaml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.yaml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、リポジトリーのルートディレクトリーにファイル名を「.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リポジトリーにWebブラウザーでアクセスする

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

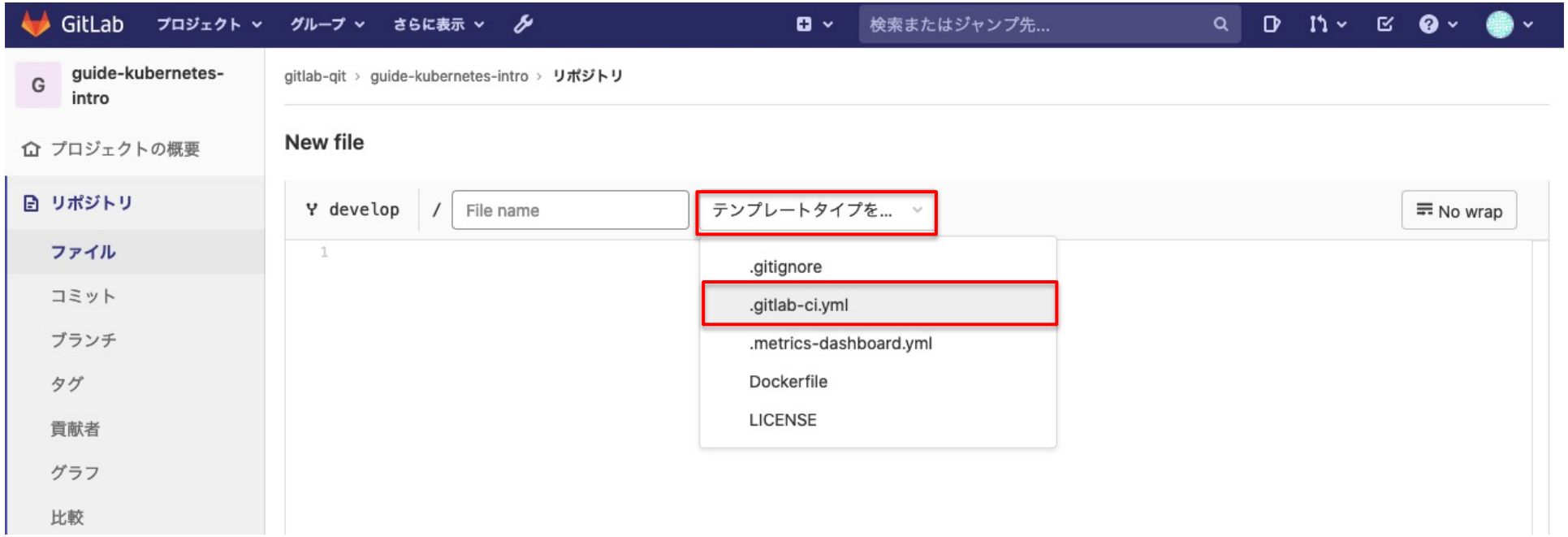
■ .gitlab-ci.yml ファイルの作成

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

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

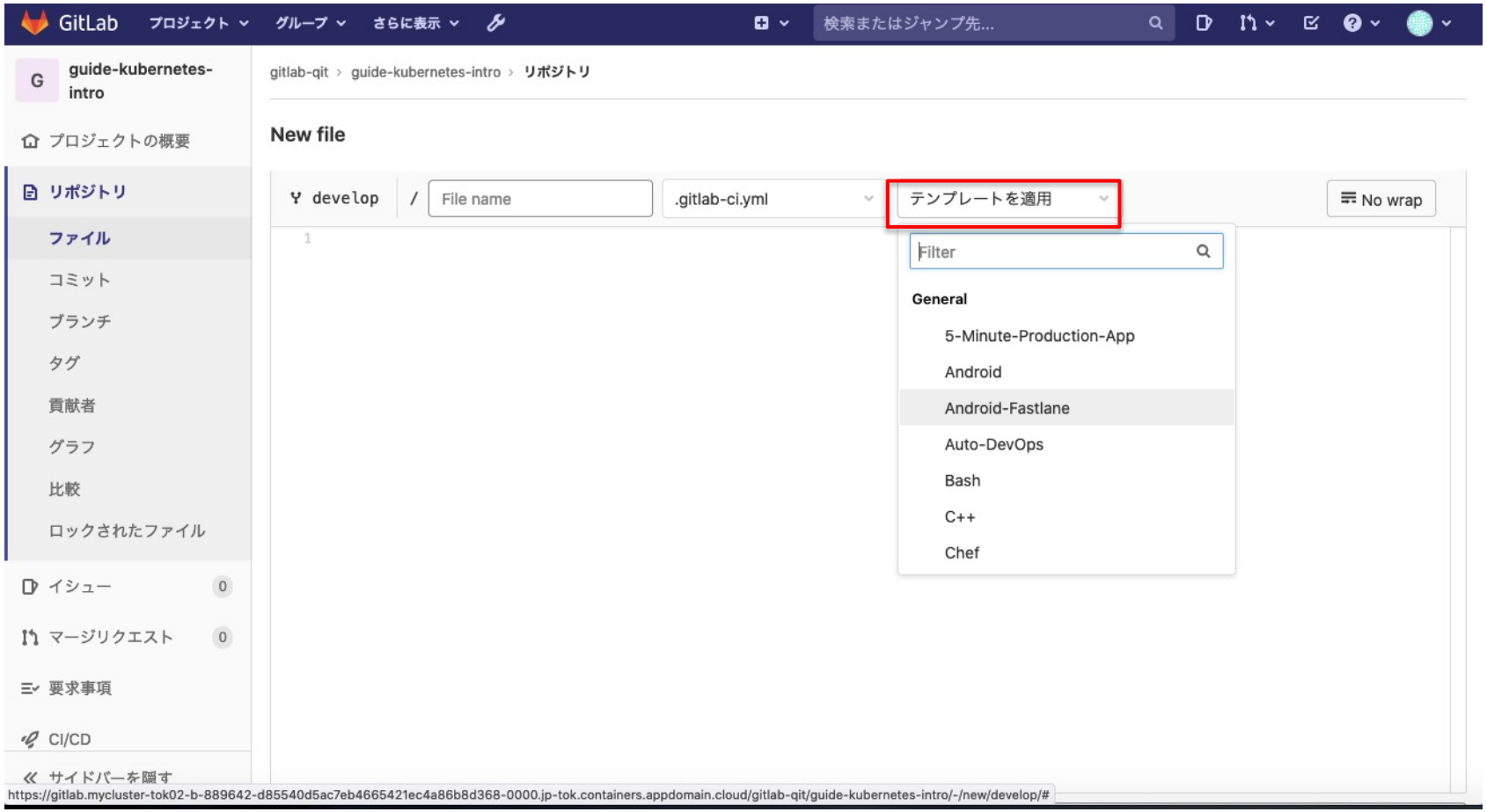
4.「テンプレートタイプを...」のリストから「.gitlab-ci.yml」をクリックする



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

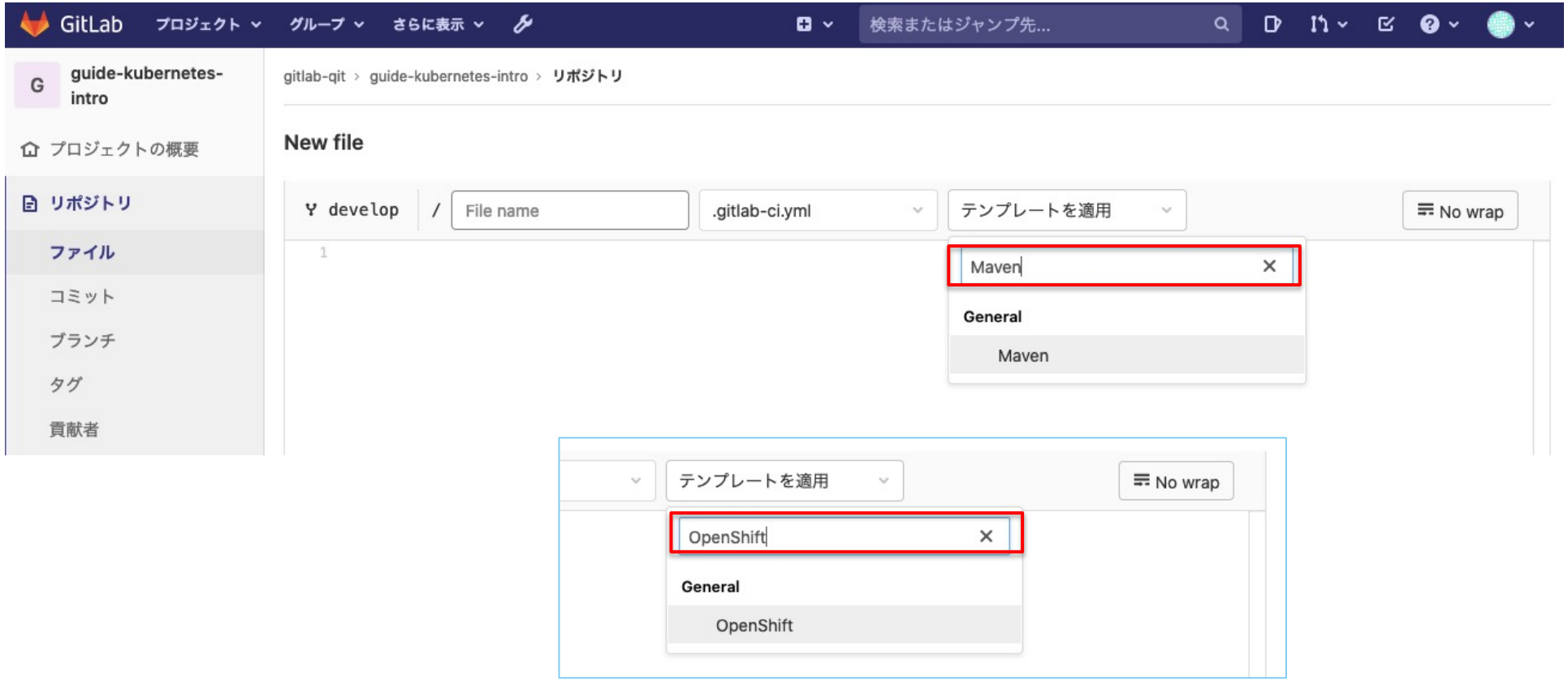
5.「テンプレートを適用」のリストから処理内容に合う項目ををクリックする



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

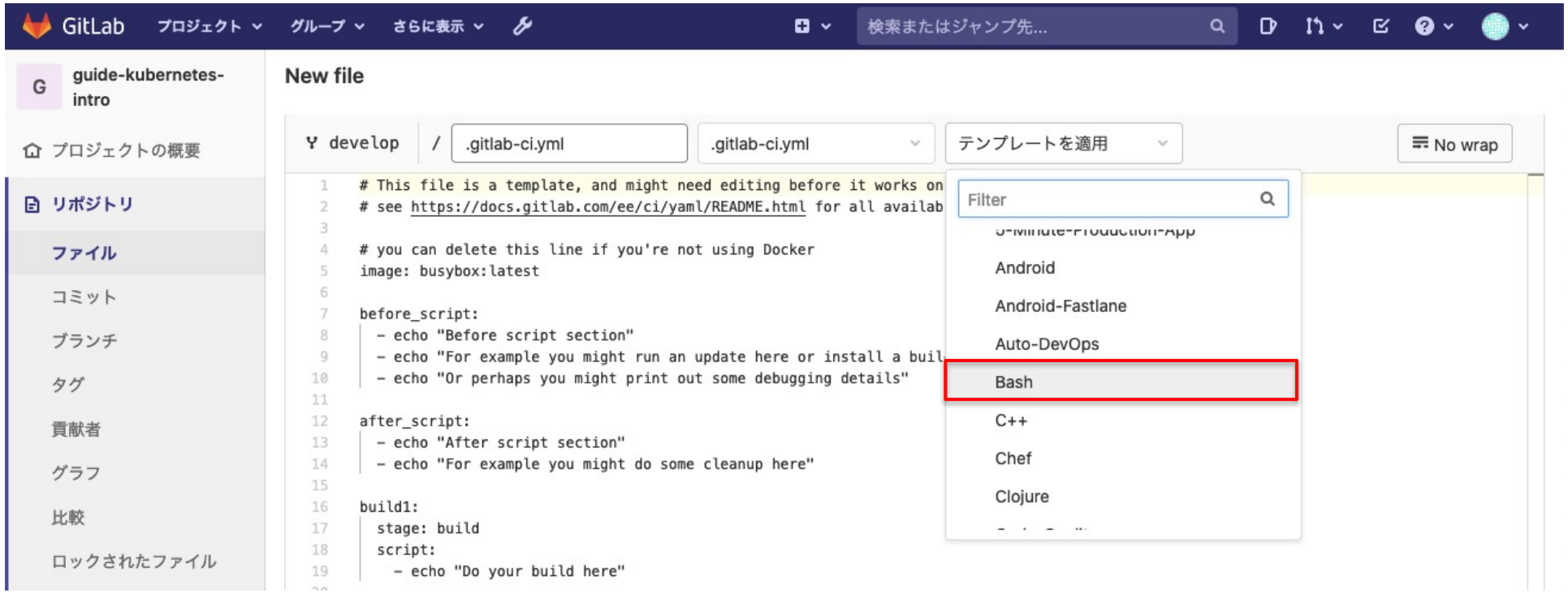
6.選択項目の上部に文字を入力すると、入力した内容でフィルターして表示される



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。



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。

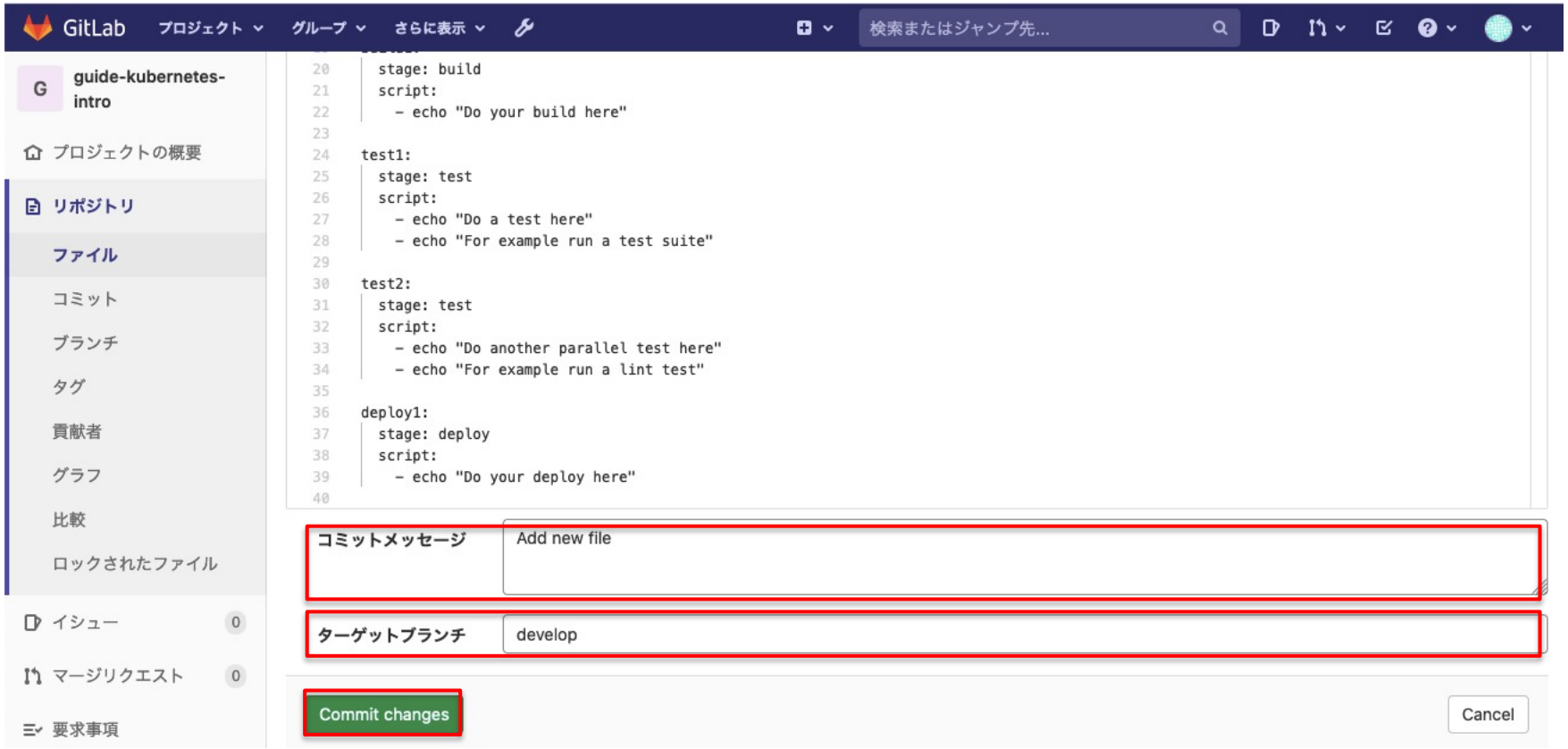
例として、GitLabに自己署名証明書を利用していて暫定的にSSL証明書の検査を無効にする場合、変数として以下の設定を追加する

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

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



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_OPS、MAVEN_CLI_OPS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

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
```

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）

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. リポジトリー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リポジトリーへのコミット**
5. 結果の確認

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

Gitリポジトリーのルートにgitlab-ci.ymlファイルを格納すると、以降そのリポジトリーにコミットが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

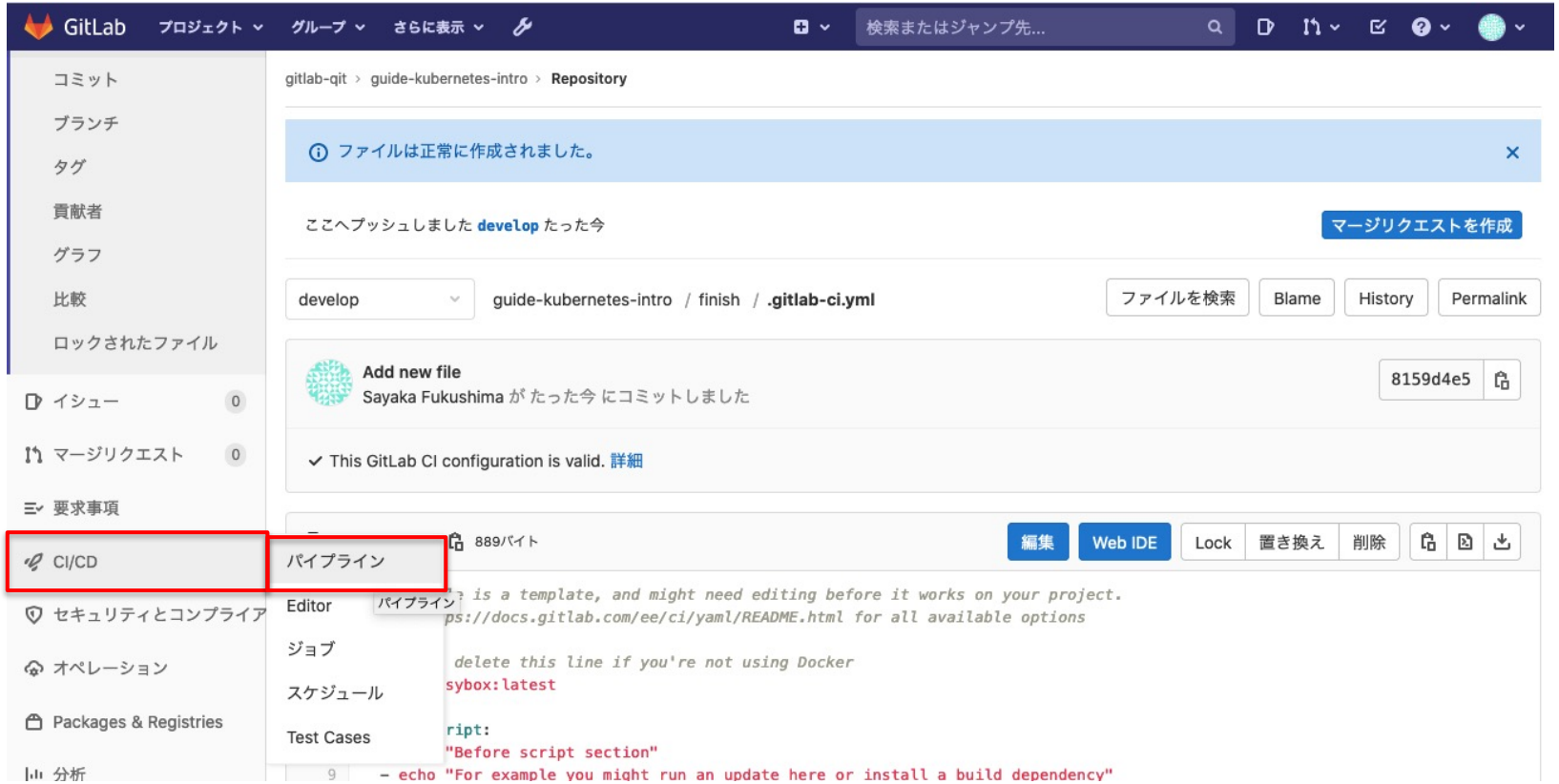
GitLab CI/CD利用の流れは以下の通り

1. リポジトリー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リポジトリーへのコミット
5. **結果の確認**

3.3. GitLab CI/CD利用ステップ – 5. 結果の確認

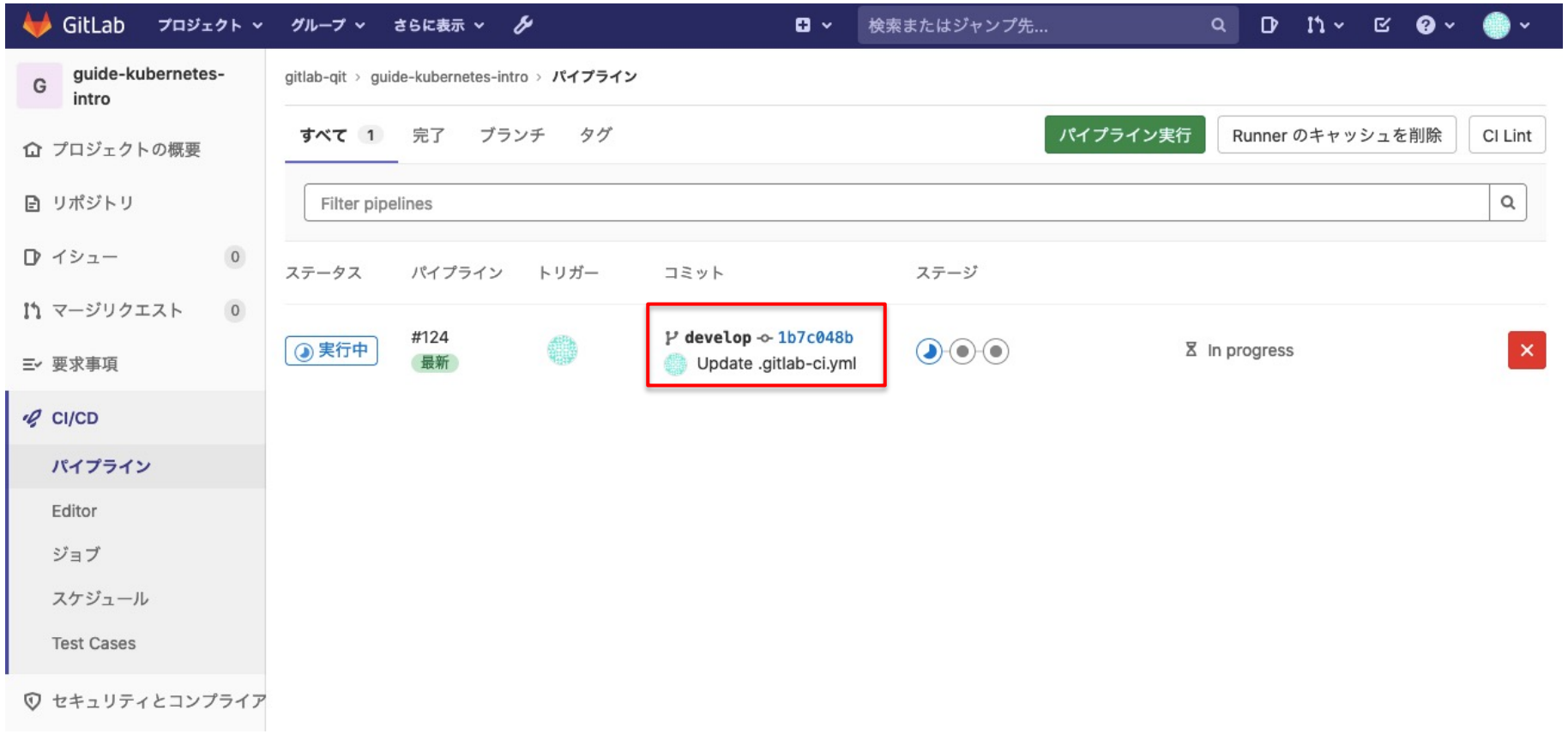
GitLab CI/CDの実行結果はGitLabのWeb UIから確認する

1. ブラウザーでgitlab-ci.ymlを格納したGitリポジトリーへアクセスする
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



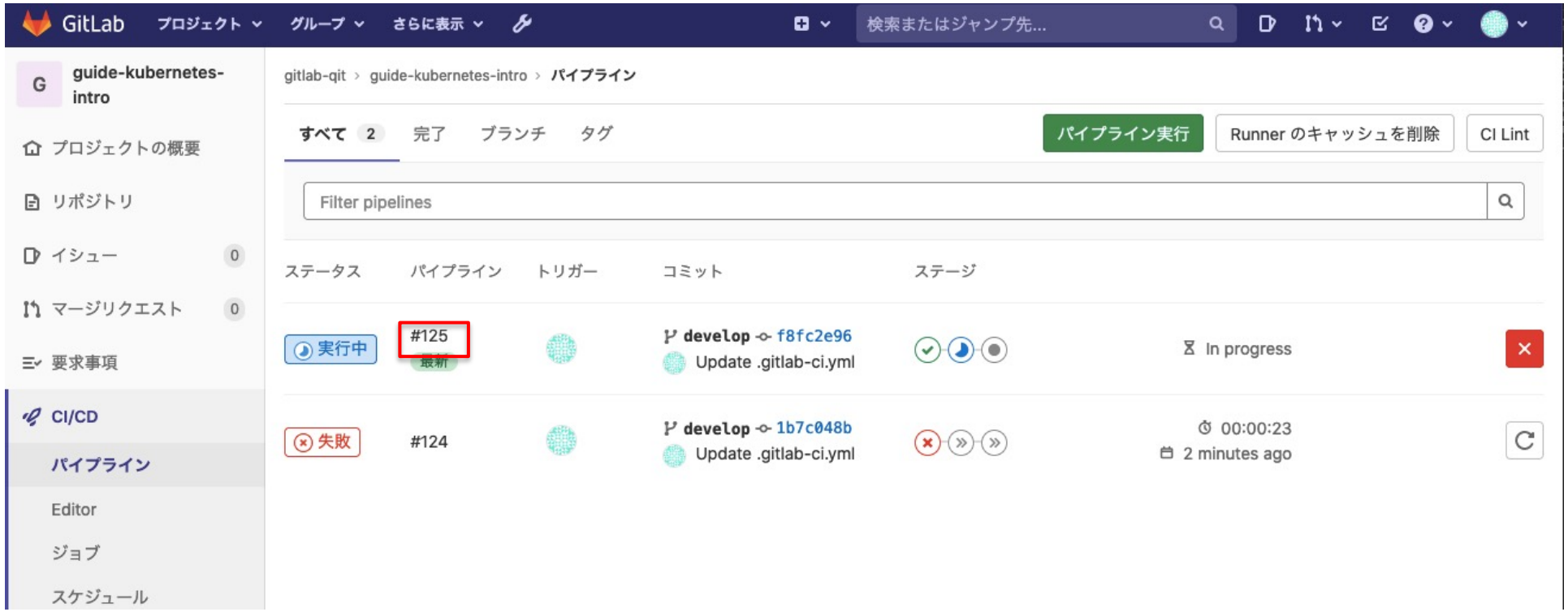
3.3. GitLab CI/CD利用ステップ – 5. 結果の確認

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



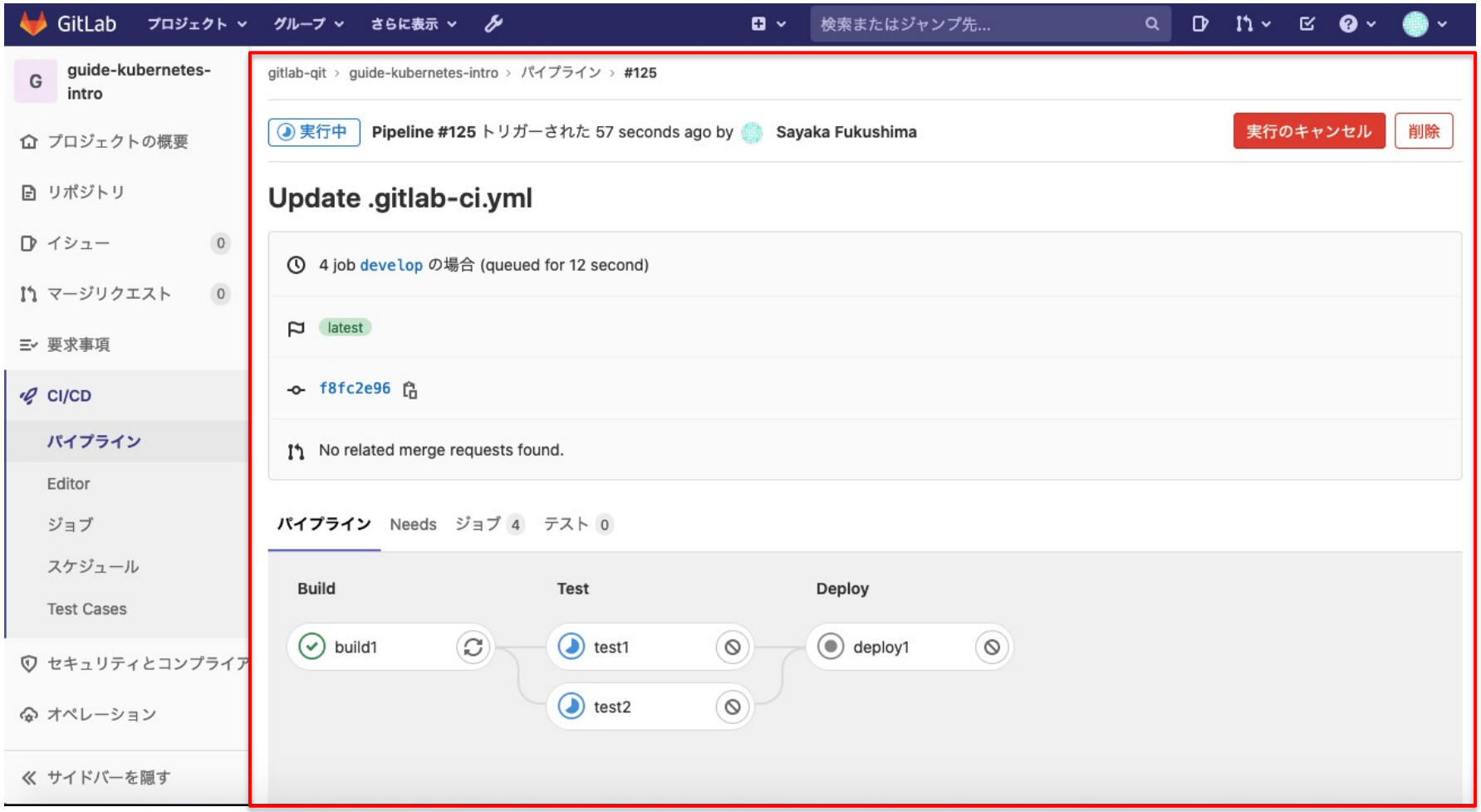
3.3. GitLab CI/CD利用ステップ – 5. 結果の確認

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



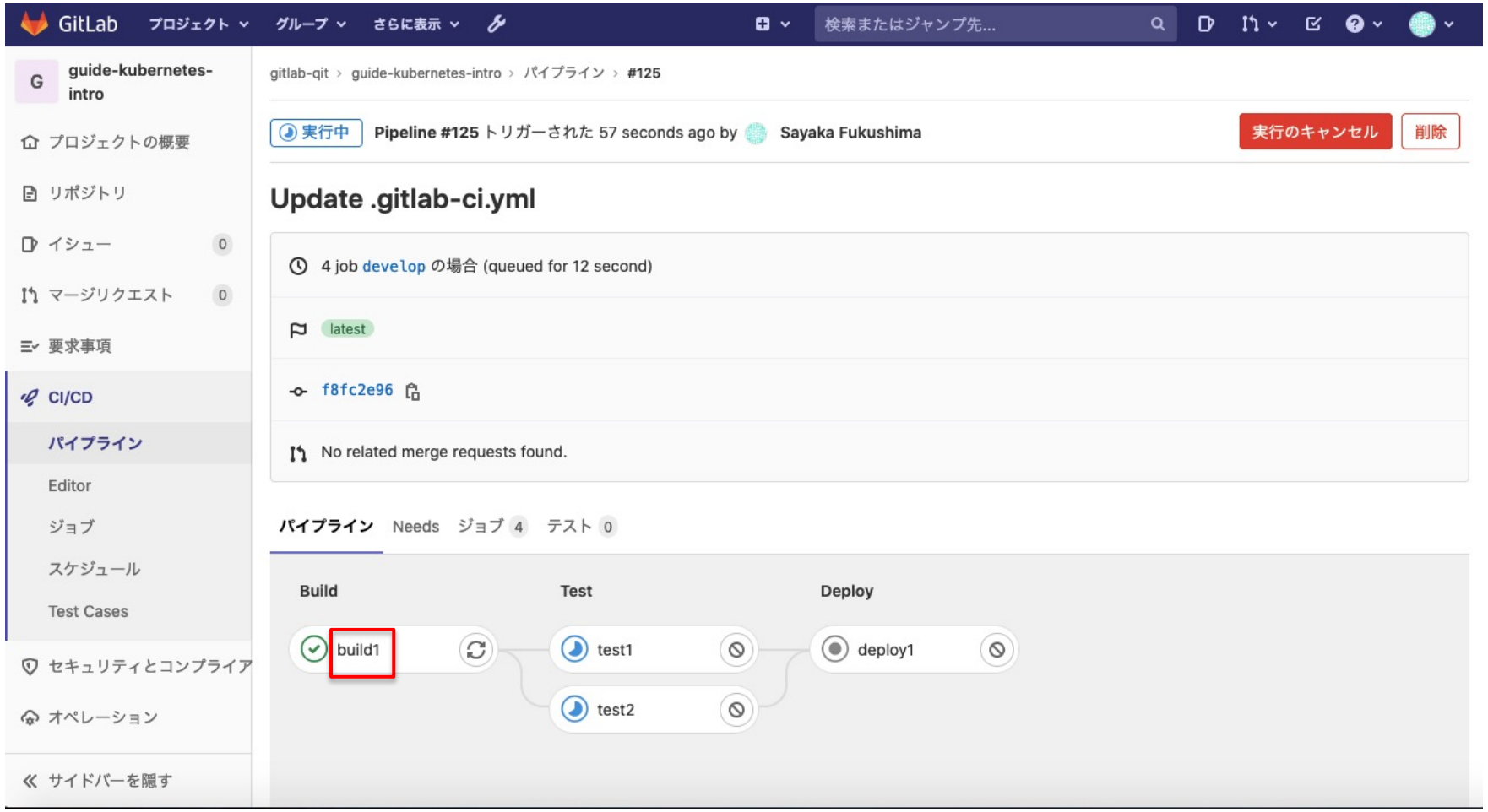
3.3. GitLab CI/CD利用ステップ – 5. 結果の確認

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



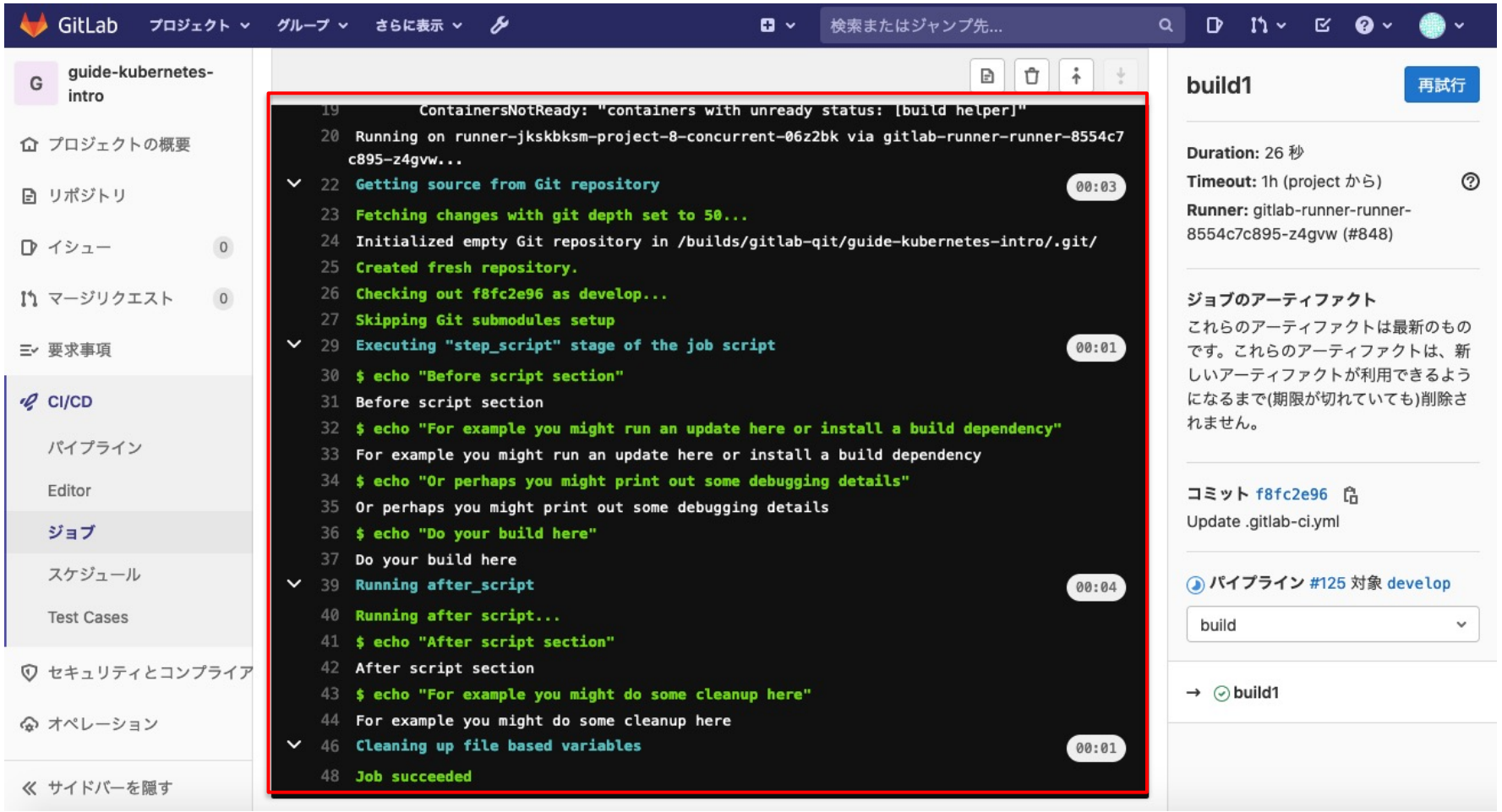
3.3. GitLab CI/CD利用ステップ – 5. 結果の確認

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



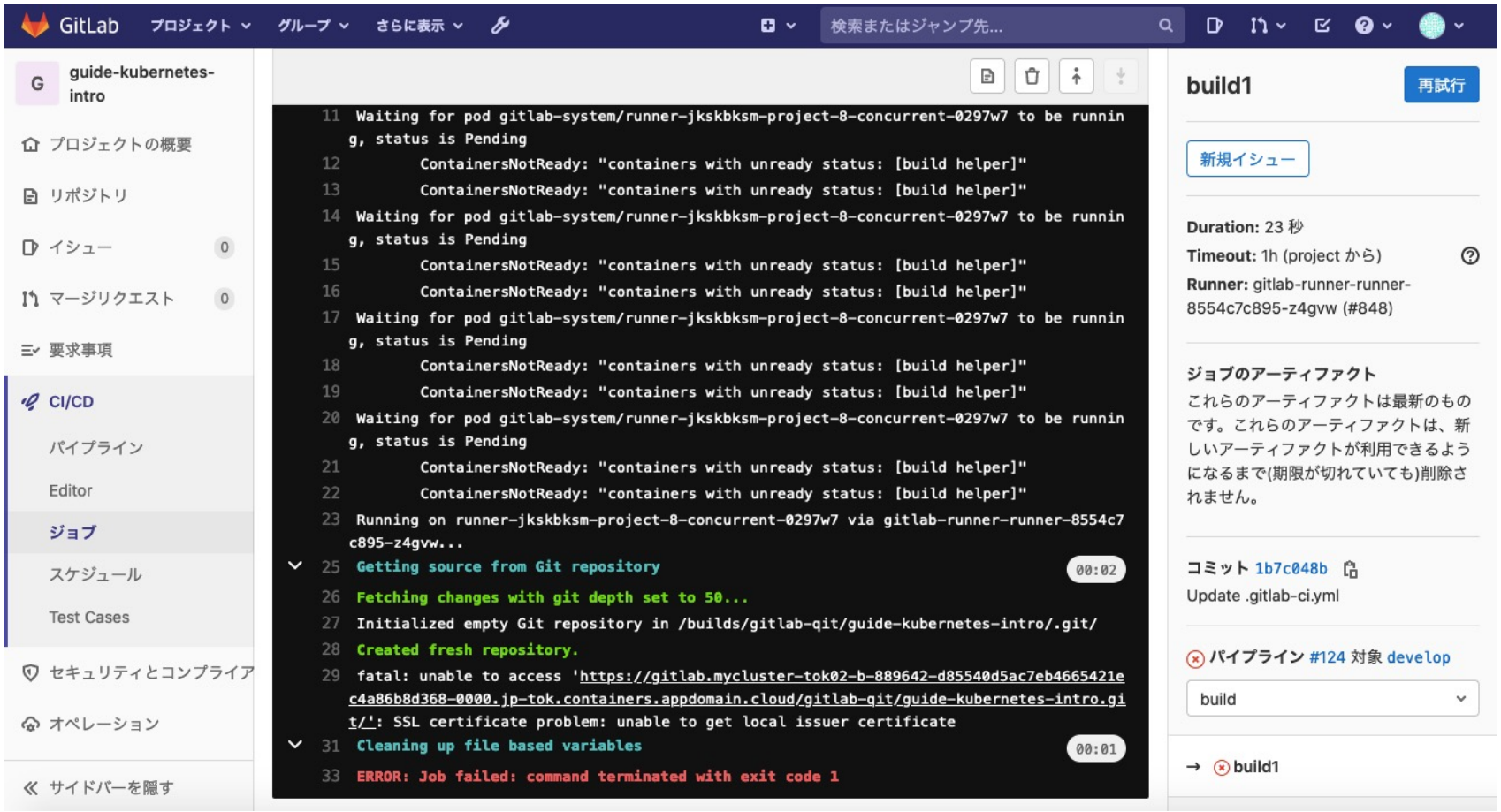
3.3. GitLab CI/CD利用ステップ – 5. 結果の確認

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



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



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リポジトリーのパスに移動（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

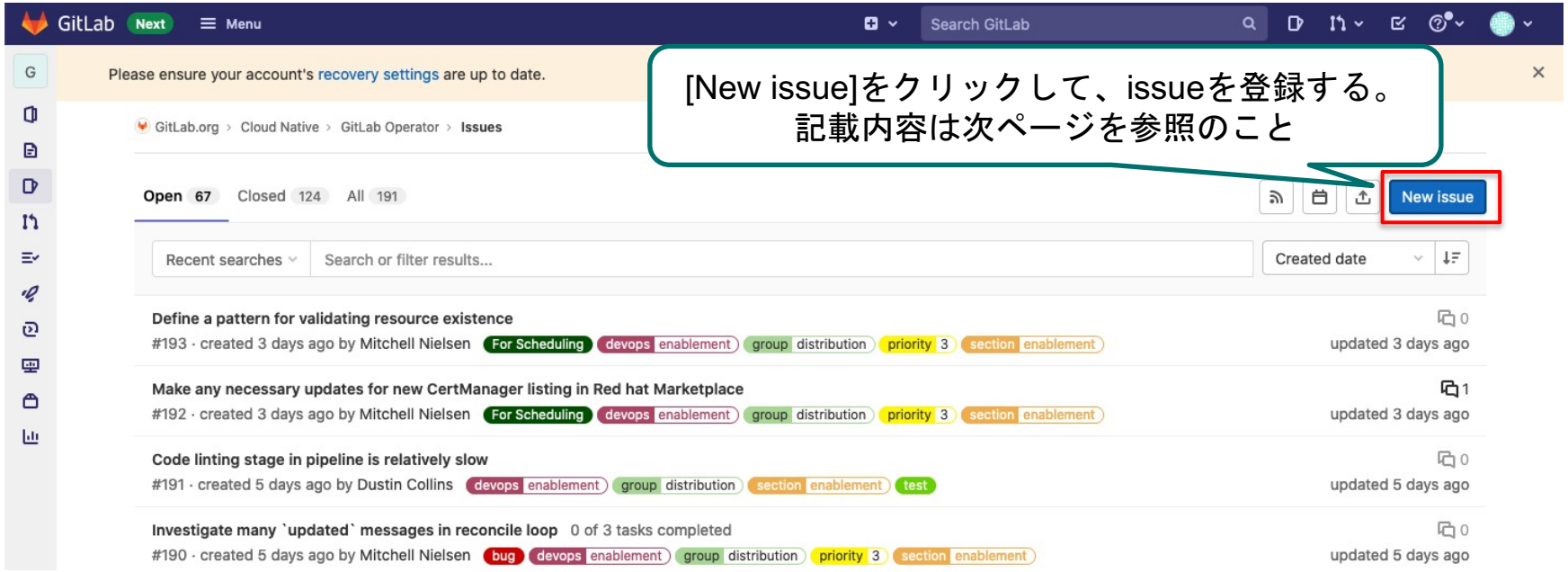
- OpenShiftクラスターにGitLab Operatorでインストールしたgitlabであれば、レジストリーの証明書は、gitlab-systemプロジェクトのroute/gitlab-registry-<ランダムな値>　のCertificateの値を利用すれば良い

- 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s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のリポジトリーに公開されている
- 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ーの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
  - リポジトリーにより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



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

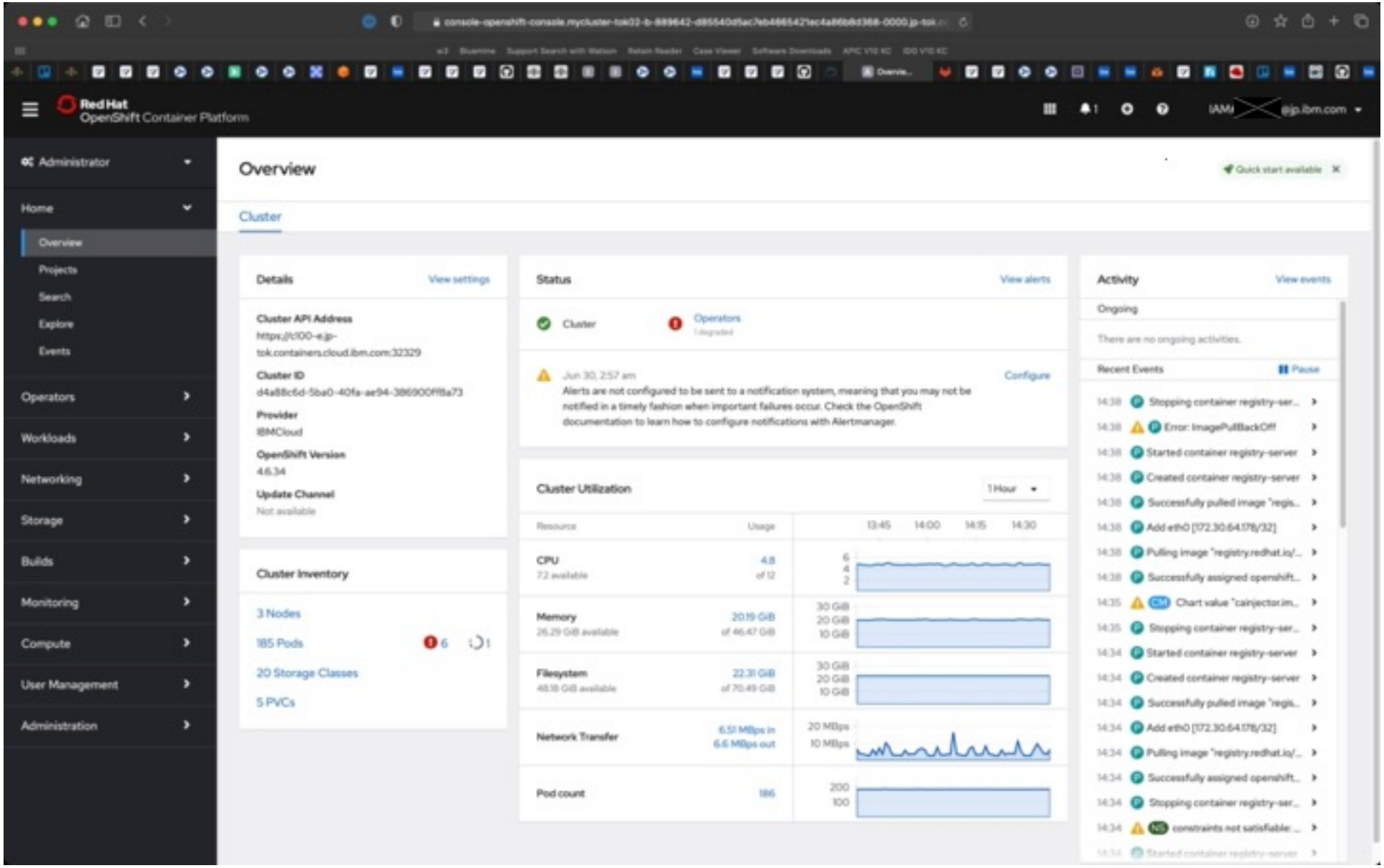
- Labelsなどその他の情報はDeveloperにより設定されると考えられるため、登録時には指定しない

Tips4. 稼働時の消費リソース例

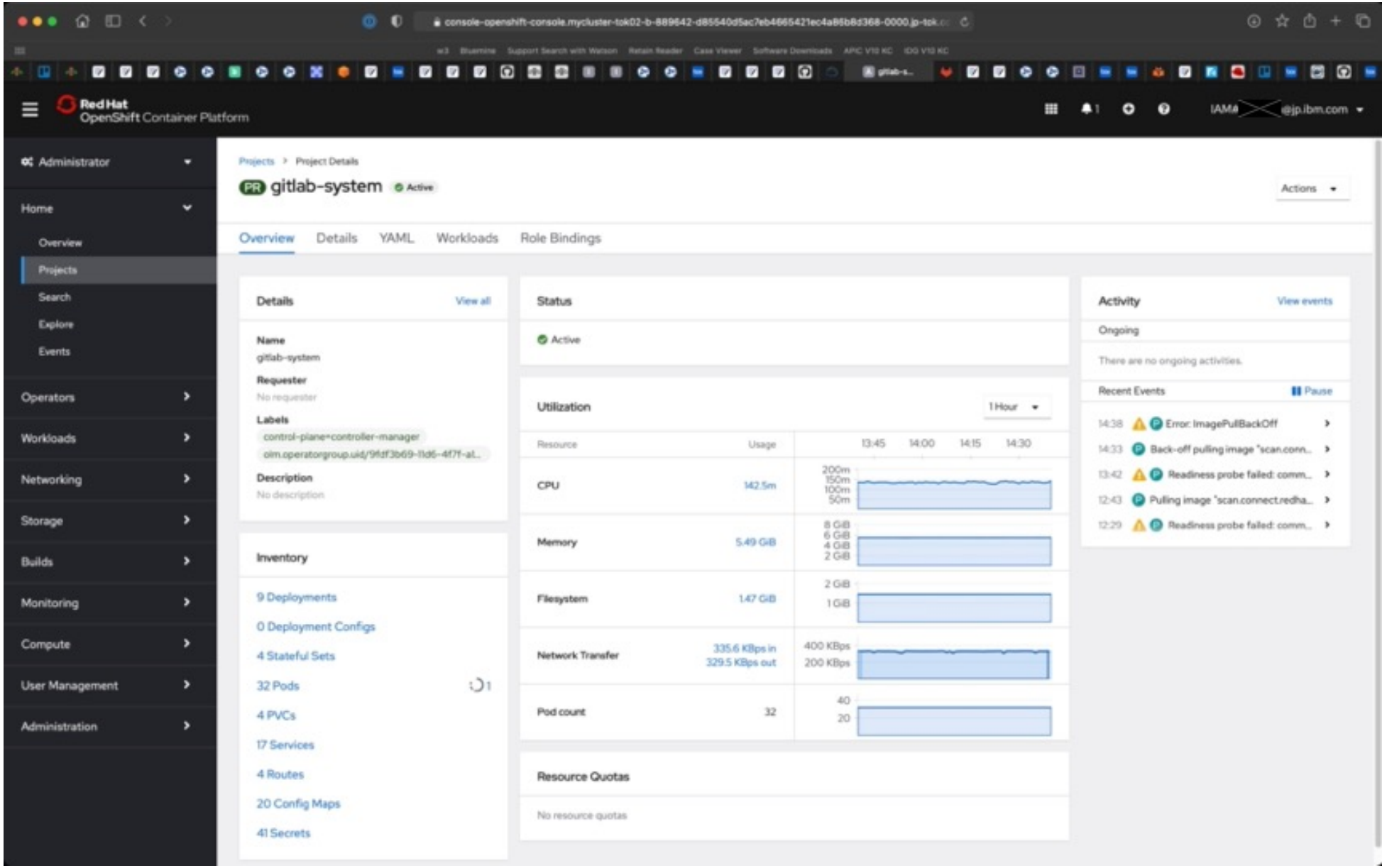
- 無負荷時の kubectl top pod -n gitlab-system

```
% 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-mbknz                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



Tips4. 稼働時の消費リソース例



ありがとうございました