GitOps-Driven Modern Database Deployment on Hitachi Unified Compute Platform with OpenShift: An Example Using OpenShift GitOps to Deploy PostgreSQL

Reference Architecture Guide

MK-SL-267-00
April 2023

© 2023 Hitachi Vantara LLC. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or by any means, electronic or mechanical, including copying and recording, or stored in a database or retrieval system for commercial purposes without the express written permission of Hitachi, Ltd., or Hitachi Vantara LLC (collectively “Hitachi”). Licensee may make copies of the Materials provided that any such copy is: (i) created as an essential step in utilization of the Software as licensed and is used in no other manner; or (ii) used for archival purposes. Licensee may not make any other copies of the Materials. “Materials” mean text, data, photographs, graphics, audio, video and documents.

Hitachi reserves the right to make changes to this Material at any time without notice and assumes no responsibility for its use. The Materials contain the most current information available at the time of publication.

Some of the features described in the Materials might not be currently available. Refer to the most recent product announcement for information about feature and product availability, or contact Hitachi Vantara LLC at https://support.hitachivantara.com/en_us/contact-us.html.

Notice: Hitachi products and services can be ordered only under the terms and conditions of the applicable Hitachi agreements. The use of Hitachi products is governed by the terms of your agreements with Hitachi Vantara LLC.

By using this software, you agree that you are responsible for:

1. Acquiring the relevant consents as may be required under local privacy laws or otherwise from authorized employees and other individuals; and
2. Verifying that your data continues to be held, retrieved, deleted, or otherwise processed in accordance with relevant laws.

Notice on Export Controls. The technical data and technology inherent in this Document may be subject to U.S. export control laws, including the U.S. Export Administration Act and its associated regulations, and may be subject to export or import regulations in other countries. Reader agrees to comply strictly with all such regulations and acknowledges that Reader has the responsibility to obtain licenses to export, re-export, or import the Document and any Compliant Products.

Hitachi and Lumada are trademarks or registered trademarks of Hitachi, Ltd., in the United States and other countries.

AIX, AS/400e, DB2, Domino, DS6000, DS8000, Enterprise Storage Server, eServer, FICON, FlashCopy, GDPS, HyperSwap, IBM, Lotus, MVS, OS/390, PowerHA, PowerPC, RS/6000, S/390, System z9, System z10, Tivoli, z/OS, z9, z10, z13, z14, z15, z16, z/VM, and z/VSE are registered trademarks or trademarks of International Business Machines Corporation.

Active Directory, ActiveX, Bing, Excel, Hyper-V, Internet Explorer, the Internet Explorer logo, Microsoft, Microsoft Edge, the Microsoft corporate logo, the Microsoft Edge logo, MS-DOS, Outlook, PowerPoint, SharePoint, Silverlight, SmartScreen, SQL Server, Visual Basic, Visual C++, Visual Studio, Windows, the Windows logo, Windows Azure, Windows PowerShell, Windows Server, the Windows start button, and Windows Vista are registered trademarks or trademarks of Microsoft Corporation. Microsoft product screen shots are reprinted with permission from Microsoft Corporation.

All other trademarks, service marks, and company names in this document or website are properties of their respective owners.

Copyright and license information for third-party and open source software used in Hitachi Vantara products can be found in the product documentation, at https://www.hitachivantara.com/en-us/company/legal.html or https://knowledge.hitachivantara.com/Documents/Open_Source_Software.

Feedback

Hitachi Vantara welcomes your feedback. Please share your thoughts by sending an email message to SolutionLab@HitachiVantara.com. To assist the routing of this message, use the paper number in the subject and the title of this white paper in the text.

Revision history

Changes	Date
Initial release	April 17, 2023

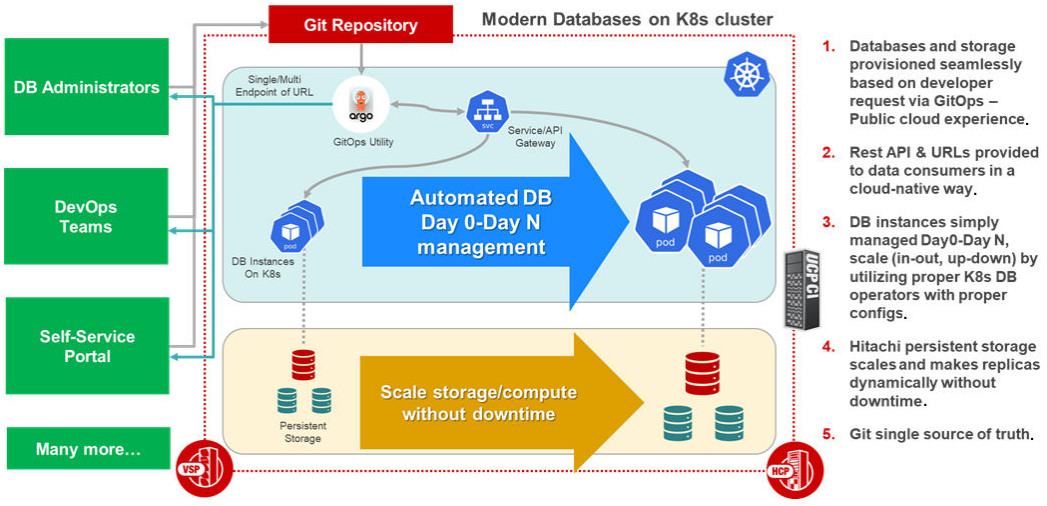
Reference Architecture Guide

Executive overview

Addressing the scalability and diversity of contemporary IT infrastructure in hybrid cloud ecosystems demands advanced automation. GitOps, a powerful framework that unifies infrastructure and application management by using Git as a centralized single source of truth for code and configurations, ensures consistency, version control, and efficient sharing across projects and environments.

Hitachi Unified Compute Platform (UCP) embraces GitOps-based solutions to optimize IT automation and management for customers, providing a powerful and efficient infrastructure that enables businesses to thrive in a competitive landscape.

As modern applications operate in complex hybrid multi-cloud environments, adopting GitOps is essential for streamlined deployment and management. Developers and platform operators need a clear, manageable process to deploy infrastructure resources, such as database instances and persistent storage volumes, using a single source of truth. This approach prevents new silos and confusion while enabling seamless cloud-like service integration, regardless of location.



Using version control, declarative infrastructure, and automation, GitOps streamlines deployment, fosters collaboration, and minimizes human error. Ultimately, this approach boosts business efficiency and continuity.

This paper demonstrates a GitOps-driven framework for deploying modern databases on a Hitachi UCP Kubernetes Solution with Red Hat OpenShift. The paper presents an example of deploying a PostgreSQL database through Kubernetes Operator and Argo CD-based OpenShift GitOps on top of an OpenShift cluster. The solution also includes the use of Persistent Volumes (PVs) to store the respective data, which are automatically created by the Hitachi Storage Plug-in for Containers on Hitachi Virtual Storage Platform (VSP). The paper describes how the GitOps framework can enable a simple and versatile continuous delivery pipeline for a modern and flexible database deployment on UCP with OpenShift.

Use case examples – databases across a vast geographical area

When managing multiple edge locations across a vast geographical area, organizations face challenges related to edge computing, hybrid cloud, limited field teams, and complex deployment and management. Real-time analytics and extensive onsite data storage are crucial in various industries and scenarios. Hitachi UCP and Openshift GitOps can address these challenges by simplifying deployment, management, and scaling of databases using the GitOps methodology.

Some real-world scenarios where Hitachi UCP and Openshift GitOps provide exceptional solutions include:

- Retail Chains: Ensuring smooth operations and quick decision-making through consistent database deployment and management across multiple locations.
- Smart City Infrastructure: Managing traffic, public transportation, and utilities with immediate data analysis and action.
- Healthcare: Efficient data access and processing for remote hospitals and clinics to improve patient care.
- Manufacturing: Real-time analytics for equipment performance, production metrics, and quality control at factories and production facilities.
- Energy and Utilities: Optimizing operations, predicting equipment failures, and managing resources efficiently for power plants and utility companies.
- Telecommunications: Monitoring network performance, managing traffic, and optimizing resources for telecom operators.

Hitachi UCP offers a reliable, flexible, and scalable infrastructure, while Argo CD ensures consistency, security, and compliance across all sites. This combination is ideal for organizations requiring real-time analytics and significant onsite data storage capabilities.

Background - GitOps simplifies modern DevOps

Modern applications are often built using a microservices architecture, where the application is broken down into small, loosely coupled, and independently deployable services. This approach allows for greater flexibility, scalability, and maintainability. However, many organizations are still practicing manual tests and deployments that can struggle to keep up with the fast-paced development and continuous evolution of these applications.

The drawbacks of manual tests and deployments

When performed repeatedly, manual tasks are susceptible to errors and often result in inconsistencies. This not only consumes valuable IT resources, but also impedes agile and efficient operations. The following are some of the issues associated with manual testing and deployment:

- Accumulation of technical issues because of inadequate code analysis
- Regression and other issues overlooked until software is in production
- Anomalies in environment configuration and slow issue resolution
- Time-consuming and risk-prone deployments

These drawbacks hinder agile operations, waste valuable IT resources, and can negatively impact the overall software development and service delivery process. By embracing automation and implementing GitOps practices, organizations can overcome these challenges and create a more efficient and reliable software development environment.

The rationale for automating manual processes

Contemporary DevOps teams are now embracing continuous integration and continuous delivery (CI/CD) that employs automation to enable continuous process and software improvement, streamlining development, testing, and deployment tasks for faster software iterations.

DevOps engineers use GitOps-based CI/CD tools such as Argo CD to automate implementation steps across multiple environments while maintaining a single source of truth. These tools provide reporting capabilities to quickly identify and address discrepancies.

Automation is essential for successful continuous delivery, making processes repeatable, auditable, and minimizing human error. This consistency allows teams to focus on higher-value tasks such as feature development and performance optimization.

Embracing automation through continuous delivery and GitOps-based CI/CD tools streamlines software development, replacing time-consuming manual tasks with efficient, automated workflows. This leads to enhanced collaboration, improved software quality, and faster time to market.

Advantages of GitOps-driven continuous delivery

- Cost Reduction

  GitOps-driven continuous delivery (CD) reduces staff costs by automating manual tasks, enabling developers and administrators to focus on adding new services and enhancing service levels, while leveraging Git as the single source of truth.

- Streamlined Workflows

  CD with GitOps promotes efficiency by automating workflows and providing a consistent version-controlled environment, ensuring tasks are completed uniformly and addressing DevOps challenges.

- Operational Confidence and Compliance

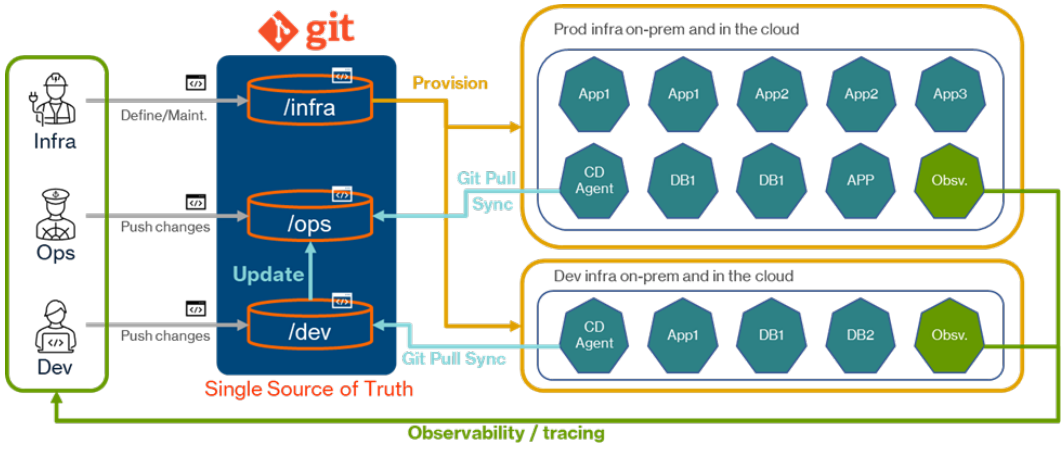
  By automating processes using GitOps, CD improves operational confidence, service levels, and regulatory compliance. Git's version control capabilities help mitigate issues related to user experience and security by providing a traceable and auditable history of changes.

- Enhanced Collaboration and Visibility

  GitOps-driven CD fosters teamwork by automating labor-intensive tasks and providing visibility into the entire infrastructure and application development lifecycle. Developers, integrators, and testers can collaborate more effectively by leveraging Git for change management and delivering software faster.

What is GitOps?

GitOps is an operational framework that simplifies and automates infrastructure and application management using Git as the single source of truth. It treats the state of infrastructure, platform, and applications as code. This allows configurations to be defined as code that is version controlled in Git, and shared and reused consistently across projects and environments. Changes to the system are made by updating the Git repository, which makes them auditable and repeatable. GitOps also supports a declarative approach to infrastructure and application management. The state is defined up front, and the system converges to that state automatically, improving reliability.



GitOps enables organizations to manage their entire infrastructure and application development lifecycle using a single, unified tool that applies the principles of version control, collaboration, compliance, and CI/CD to infrastructure automation. By using Git as the single source of truth to manage all infrastructure files, GitOps simplifies and automates infrastructure and application management. To perform GitOps operations, infrastructure files must be checked in to Git or forked from a public repository. With GitOps, organizations can manage their infrastructure and application development lifecycle using a single, unified tool that provides version control, collaboration, compliance, and CI/CD capabilities to infrastructure automation.

What are OpenShift GitOps and Argo CD?

OpenShift GitOps is a continuous delivery tool for managing Kubernetes applications that uses the GitOps style and is built on top of Argo CD and other tooling. It operates declaratively, ensuring that the application always runs in its desired state.

Argo CD is a component of the Argo project suite of products that also includes Argo Workflow, Argo Rollout, and Argo Event. These products help solve specific problems in the agile development process and make Kubernetes application delivery scalable and secure. By managing applications and infrastructure as code, developers can automate the deployment process and enable faster, more reliable delivery.

Use GitOps to simplify modern DevOps

GitOps addresses the challenges of manual testing and deployment processes, replacing them with efficient, automated workflows that streamline software development, enhance collaboration, and improve software quality. It simplifies and automates infrastructure and application management using Git as the single source of truth, providing version control, collaboration, compliance, and CI/CD capabilities to infrastructure automation.

With tools like OpenShift GitOps and Argo CD, developers can manage their applications and infrastructure as code, automating the deployment process and enabling faster, more reliable delivery, while promoting efficiency, operational confidence, and regulatory compliance.

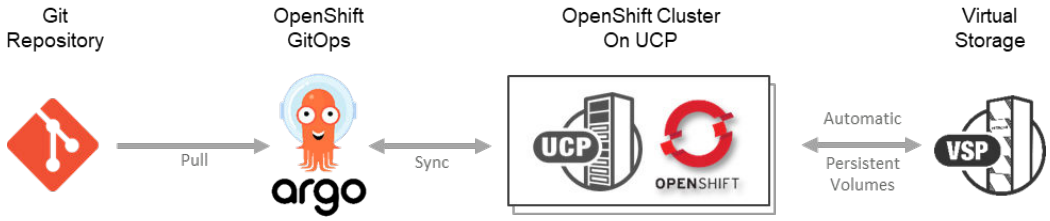
Kubernetes operators

A Kubernetes Operator extends Kubernetes functionality, managing the lifecycle of complex, stateful applications. It automates and simplifies tasks such as deployment, scaling, and updates by encoding domain knowledge into software, allowing Kubernetes to treat applications as native components. See https://kubernetes.io/docs/concepts/extend-kubernetes/operator/ for more information.

Argo CD passes configurations to Kubernetes operators by applying custom resources, which contain the desired state and configuration details of the application. The operator's custom controller detects changes in these resources and takes corrective actions to reconcile the application's actual state with the desired state, ensuring seamless management of the application within the Kubernetes environment.

The GitOps-Driven framework on UCP with OpenShift

This paper introduces a versatile GitOps-driven framework that uses Argo CD (OpenShift GitOps), an open-source Kubernetes operator for databases (with CrunchyData Postgres Operator as an example), GitHub, Hitachi Virtual Storage Platform (VSP), and Hitachi Storage Plug-in for Containers to automate and simplify the database deployment process. PostgreSQL is selected because it is one of the most widely used databases in contemporary applications and boasts a substantial user base.



The framework uses Argo CD to retrieve configurations from a Git repository, facilitating the deployment, scaling, and modification of the PostgreSQL database through the CrunchyData PostgreSQL operator. Simultaneously, the underlying Storage Plug-in for Containers and VSP automatically allocate and deliver persistent storage volumes for the database, guaranteeing a dynamic, automated, and streamlined deployment process.

Additionally, this versatile framework can be adapted to accommodate other databases operating on Kubernetes, establishing itself as a valuable asset for organizations that support modern data applications with a GitOps methodology.

Crunchy Data's Postgres Operator

Crunchy Data's Postgres Operator is an open-source project, and you do not need a subscription to use its core functionality. You can access the source code and documentation on their GitHub repository (https://github.com/CrunchyData/postgres-operator) and use it freely to experiment, deploy, manage, and scale PostgreSQL clusters on Kubernetes.

While the Postgres Operator simplifies PostgreSQL cluster management on Kubernetes, by integrating Argo CD, organizations can benefit from the GitOps approach. Continuous delivery, streamlined cluster management, enhanced security, and seamless tool integration make PostgreSQL deployments more efficient, consistent, scalable, and secure.

Hitachi Unified Compute Platform Kubernetes solution and Hitachi Virtual Storage Platform

To thrive in today's digital economy, you need a more advanced and reliable IT infrastructure that can handle processes, and data faster and more efficiently in a secure manner. By leveraging such a sophisticated infrastructure including Hitachi Unified Compute Platform (UCP) solution backed by Hitachi Virtual Storage Platform (VSP), you can accelerate business decisions as well as store/fetch respective data on/from the fastest and the most secure storage.

The following are some of the advantages of using Hitachi UCP Kubernetes Solution backed by VSP storage systems:

- Improve scalability for growth
- Maximize data reduction technology
- Consolidate multiple workloads with confidence
- Reduce disruption for higher performance
- Achieve consistent long-term performance

Hitachi Storage Plug-in for Containers

Hitachi Storage Plug-in for Containers integrates Kubernetes with Hitachi storage systems using the Container Storage Interface (CSI). It enables dynamic provisioning of the virtual storage software block system, where containers are used to integrate virtual storage software blocks into Kubernetes-based OpenShift container platform clusters and support both Kubernetes and Red Hat OpenShift container platforms. Storage Plug-in for Containers provides persistent volumes from Hitachi storage systems.

Project configuration

This example configuration is deployed and run on top of an OpenShift cluster (version 4.12) configured with 3 × virtual master nodes and 3 × worker nodes. One of the worker nodes is a physical node (hh02-worker-3) connected to Hitachi Virtual Storage Platform (VSP) using an Emulex HBA on a Fibre Channel network.

Hardware configuration

The following table lists the main hardware components.

Hardware	Type	Configuration	Version/Model	Quantity
Master Node	Virtual	4 × vCPUs 16 GB Memory 120 GB vDisk	VM version 19	3
Worker Node	Virtual		VM version 19	2
Worker Node	Physical	Hitachi Advanced Server DS120 128 GB Memory 1 × Emulex HBA	DS120	1
Hitachi Virtual Storage Platform	Storage	165 TB Physical Capacity	VSP 5600H	1

The physical worker node “hh02-worker-3” was connected to VSP 5600H using an Emulex HBA.

Software configuration

The following table lists the main software components.

Software	Version
Red Hat OpenShift	4.12
Hitachi Storage Plug-in for Containers	3.11
Red Hat OpenShift GitOps	1.7.2
Argo CD	2.5.1
Crunchy Postgres Operator	5.0
Hitachi Storage Virtualization Operating System RF	90-08-41

Project details

Assuming you have access to a functioning OpenShift cluster with internet access, this paper demonstrates deploying a PostgreSQL database using GitOps methodology. These are the high-level steps:

1. Install and configure Argo CD.
   a. Manually install Argo CD on the OpenShift cluster.
   b. Retrieve the Argo CD admin password.
   c. Log in to the Argo CD Web UI and configure the settings.
2. Prepare and deploy PostgreSQL by following these steps:
   a. Create a repository and connect Argo CD to the repository.
   b. Update the Postgres definition file that Crunchy Data's Postgres Operator will use.
   c. Create an Argo CD project.
   d. Understand the Argo CD application concept - this is a requirement.
      - An Argo CD application is a Kubernetes custom resource that represents a group of Kubernetes resources deployed to a cluster.
   e. Deploy the Postgres Operator as an Argo CD application.
      - The Postgres Operator is a utility that, once installed, allows DevOps teams to deploy and manage Postgres databases within the Kubernetes cluster.
   f. Create an example Postgres Cluster as an Argo CD application.
      - This example demonstrates creating a Postgres Cluster using the Argo CD application concept. It simulates a real-world scenario where DevOps teams create/request databases for their services using a GitOps methodology.
   g. Validate the deployment.
3. Monitor the deployment status using the corresponding OpenShift commands.

Install Argo CD (OpenShift GitOps)

You can Install Argo CD from the command line or by installing the OpenShift GitOps operator available on Red Hat OperatorHub. The Red Hat OpenShift GitOps operator was installed for this example.

Run the following commands to complete the initial steps for a manual installation of Argo CD in your OpenShift cluster.

```
oc create namespace argocd
```


```
oc get namespaces | grep argocd
```


```
oc  apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml
```

```
[ocpinstall@hh02-ocp-admin-ws ~]$ oc apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml
customresourcedefinition.apiextensions.k8s.io/applications.argoproj.io created
customresourcedefinition.apiextensions.k8s.io/applicationsets.argoproj.io created
customresourcedefinition.apiextensions.k8s.io/appprojects.argoproj.io created
serviceaccount/argocd-application-controller created
serviceaccount/argocd-applicationset-controller created
serviceaccount/argocd-dex-server created
serviceaccount/argocd-notifications-controller created
serviceaccount/argocd-redis created
serviceaccount/argocd-repo-server created
serviceaccount/argocd-server created
role.rbac.authorization.k8s.io/argocd-application-controller created
role.rbac.authorization.k8s.io/argocd-applicationset-controller created
role.rbac.authorization.k8s.io/argocd-dex-server created
role.rbac.authorization.k8s.io/argocd-notifications-controller created
role.rbac.authorization.k8s.io/argocd-server created
clusterrole.rbac.authorization.k8s.io/argocd-application-controller created
clusterrole.rbac.authorization.k8s.io/argocd-server created
rolebinding.rbac.authorization.k8s.io/argocd-application-controller created
rolebinding.rbac.authorization.k8s.io/argocd-applicationset-controller created
rolebinding.rbac.authorization.k8s.io/argocd-dex-server created
rolebinding.rbac.authorization.k8s.io/argocd-notifications-controller created
rolebinding.rbac.authorization.k8s.io/argocd-redis created
rolebinding.rbac.authorization.k8s.io/argocd-server created
clusterrolebinding.rbac.authorization.k8s.io/argocd-application-controller created
clusterrolebinding.rbac.authorization.k8s.io/argocd-server created
configmap/argocd-cm created
configmap/argocd-cmd-params-cm created
configmap/argocd-gpg-keys-cm created
configmap/argocd-notifications-cm created
configmap/argocd-rbac-cm created
configmap/argocd-ssh-known-hosts-cm created
configmap/argocd-tls-certs-cm created
secret/argocd-notifications-secret created
secret/argocd-secret created
service/argocd-applicationset-controller created
service/argocd-dex-server created
service/argocd-metrics created
service/argocd-notifications-controller-metrics created
service/argocd-redis created
service/argocd-repo-server created
service/argocd-server created
service/argocd-server-metrics created
deployment.apps/argocd-applicationset-controller created
deployment.apps/argocd-dex-server created
deployment.apps/argocd-notifications-controller created
deployment.apps/argocd-redis created
deployment.apps/argocd-repo-server created
deployment.apps/argocd-server created
statefulset.apps/argocd-application-controller created
networkpolicy.networking.k8s.io/argocd-application-controller-network-policy created
networkpolicy.networking.k8s.io/argocd-applicationset-controller-network-policy created
networkpolicy.networking.k8s.io/argocd-dex-server-network-policy created
networkpolicy.networking.k8s.io/argocd-notifications-controller-network-policy created
networkpolicy.networking.k8s.io/argocd-redis-network-policy created
networkpolicy.networking.k8s.io/argocd-repo-server-network-policy created
networkpolicy.networking.k8s.io/argocd-server-network-policy created
[ocpinstall@hh02-ocp-admin-ws ~]$
```

```
oc get pods -n argocd -w
```

```
[ocpinstall@hh02-ocp-admin-ws ~]$ oc get pods -n argocd -w
NAME                                READY   STATUS    RESTARTS   AGE
argocd-application-controller-0     1/1     Running   0          3m11s
```

```
oc get svc -n argocd
```

```
[ocpinstall@hh02-ocp-admin-ws ~]$ oc get svc -n argocd
NAME                                      TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)                      AGE
argocd-applicationset-controller          ClusterIP   172.30.179.248   <none>        7000/TCP,8080/TCP            4m24s
argocd-dex-server                         ClusterIP   172.30.203.145   <none>        5556/TCP,5557/TCP,5558/TCP   4m24s
argocd-metrics                            ClusterIP   172.30.30.179    <none>        8082/TCP                     4m24s
argocd-notifications-controller-metrics   ClusterIP   172.30.233.152   <none>        9001/TCP                     4m24s
argocd-redis                              ClusterIP   172.30.13.166    <none>        6379/TCP                     4m24s
argocd-repo-server                        ClusterIP   172.30.29.222    <none>        8081/TCP,8084/TCP            4m24s
argocd-server                             ClusterIP   172.30.218.56    <none>        80/TCP,443/TCP               4m24s
argocd-server-metrics                     ClusterIP   172.30.100.170   <none>        8083/TCP                     4m24s
[ocpinstall@hh02-ocp-admin-ws ~]$
```

If you chose to install manually, additional steps are required to finish the installation, access the Argo CD UI, and expose the Argo CD service.

See the *Argo CD Getting Started Guide* at https://argo-cd.readthedocs.io/en/stable/getting_started/ for more details.

Note: The default Argo CD user login is admin. The initial password for the admin account is auto-generated and stored as clear text in the field password as a secret named argo-initial-admin-secret in the Argo CD installation namespace.

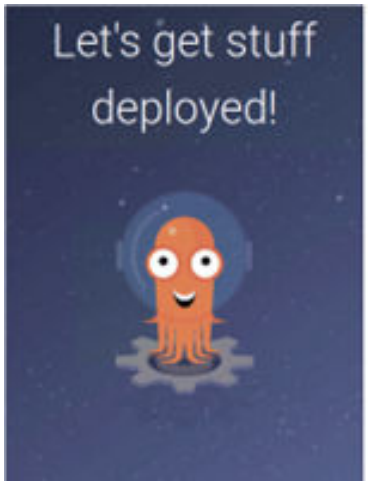
For example, the following command shows the admin password for the OpenShift GitOps operator that includes the Argo CD instance. There are multiple ways to run a command to show the Argo CD admin password.

```
argoPass=$(oc get secret/openshift-gitops-cluster -n openshift-gitops -o jsonpath='
{.data.admin\.password}' | based 64 -d)
echo $argoPass
```

The password can also be retrieved by running the following command (the namespace that Argo CD installed on it is argocd):

```
oc -n argocd get secret argocd-initial-admin-secret -o jsonpath='{.data.admin
\.password}' | based 64 -d); echo
```

To complete the initial configuration of Argo CD, log in to its Web UI using the admin username and the initial password that you acquired earlier.



After logging in, you can deploy and manage applications on your OpenShift cluster through the Argo CD Web UI.

Configure Argo CD

The stepping stone for configuring Argo CD is connecting to a repository. For example, you can either create a Git repository and then check in all your files on that repository or you can fork one of the public repositories to your Git repository.

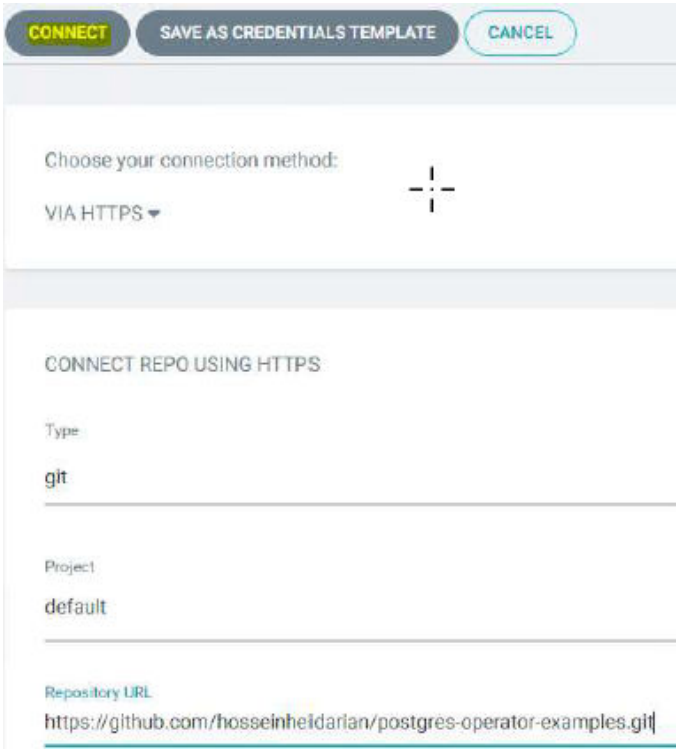
See *Getting started with GitHub* at https://docs.github.com/en/get-started for more information.

Create a repository

For this project, the Crunchy Postgres Operator was forked as a public repository on GitHub. See https://github.com/CrunchyData/postgres-operator-examples for more information.

To add a repository to Argo CD, complete the following steps:

1. Click Setting from the navigation bar on the left.
2. Click Repositories, and then click + CONNECT REPO.
3. Choose your connection method, for example VIA HTTPS.
4. Provide all the necessary information, and then click CONNECT.



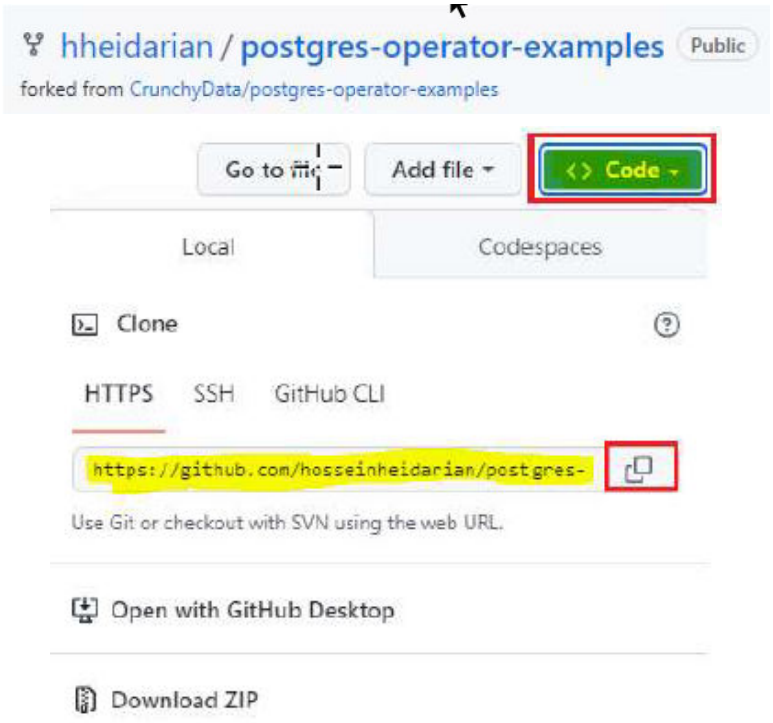
The Git repo is added to Argo CD repository.

(Optional) Cloning to a local repository

If you want to follow the exact steps in this paper, we recommend cloning or forking the Crunchy Postgres Operator in your own Git repository, either locally or in the cloud. This will allow you to modify the parameters in the manifest files as needed for your specific use case.

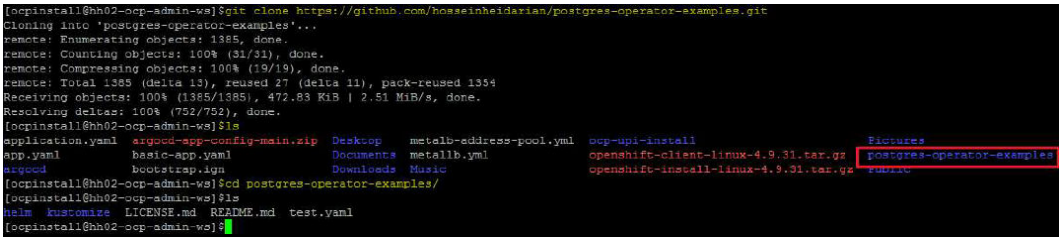
Note: After cloning or forking, remember to connect Argo CD to your repository for a successful deployment.

For this scenario, a fork was created from the Crunchy Postgres Operator on github to the github repository and that repository was cloned to the local OpenShift environment.



Also, this repository was added to the Argo CD repository as well as being cloned to the local OpenShift environment.

```
git clone https://github.com/hosseinheidarian/postgres-operator-examples.git
```



Create a project

Argo CD uses projects to logically categorize applications. To create a project, complete the following steps:

1. Click Setting.
2. Select Projects.
3. Click +NEW PROJECT.
4. Enter a name and description, and then click CREATE.
5. Add your Git repo to Source Repositories and add the OpenShift cluster (Kubernetes cluster) to Destination.

The default project is used for this example.

Understand the Argo CD application concept - this is a requirement.

In Argo CD, an application (Application CRD) is a Kubernetes custom resource that represents a group of Kubernetes resources deployed to a cluster, usually defined in a Git repository. It serves as a central unit for managing and deploying those resources. An "Argo CD application" resource specifies the desired state (declarative) of the resources and their configuration, as well as the Git repository location and the path where the resource manifests are stored. Argo CD organizes applications into projects for logical categorization.

Often, Argo CD applications have dependencies on other Argo CD applications, because one application might rely on services, resources, or configurations provided by another. Typically, an application will have multiple services and some of those services would require a database instance. For instance, a web application using data from a database requires the database to be deployed before it can be used by Argo CD applications.

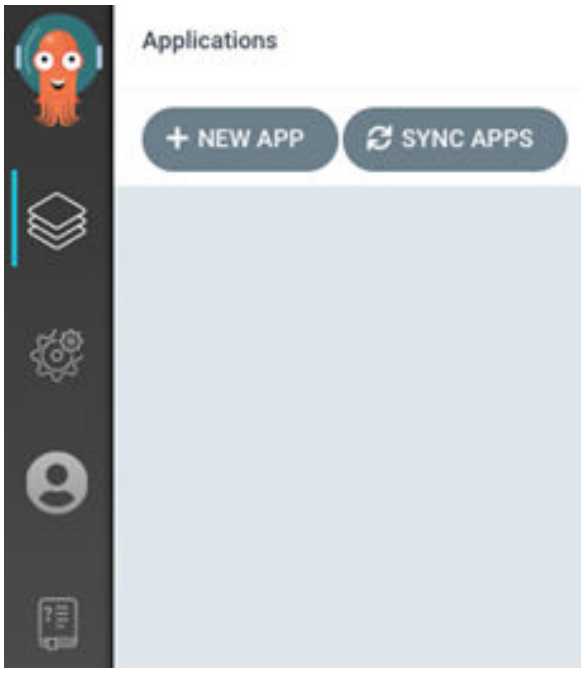
In our project, we deploy "Crunchy Postgres Operator" and a "Postgres database" as two separate "Argo CD applications" in the correct sequence. First, we deploy the Crunchy Postgres Operator, which equips the targeted Kubernetes cluster with the ability to manage and deploy Postgres databases and their associated resources.

After the Crunchy Postgres Operator is deployed, the Kubernetes cluster can deploy and manage Postgres databases. Then, we deploy a Postgres database as another "Argo CD application," ensuring proper dependency management and sequencing.

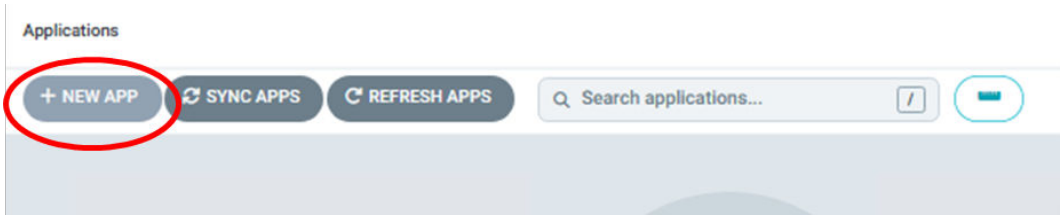
Before deploying an application, you need to know what the prerequisites and dependencies are, and how you can customize the deployment process.

Application deployment

1. To deploy an application, click Applications from the navigation bar on the left.



2. Click + NEW APP.



- Provide a name for the application.
- Select the project that you created earlier or select the default if you have not created a project yet.
- Select application SYNC POLICY OPTIONS.
- Select PRUNE PROPAGATION POLICY.
- Under SOURCE:
  - Enter the repository URL.
  - Select Revision.
  - Enter the path to the application definition files.
- Under DESTINATION:
  - Enter the cluster URL. For example, `https://kubernetes.default.svc`.
  - Enter the namespace that the application is going to be deployed on.

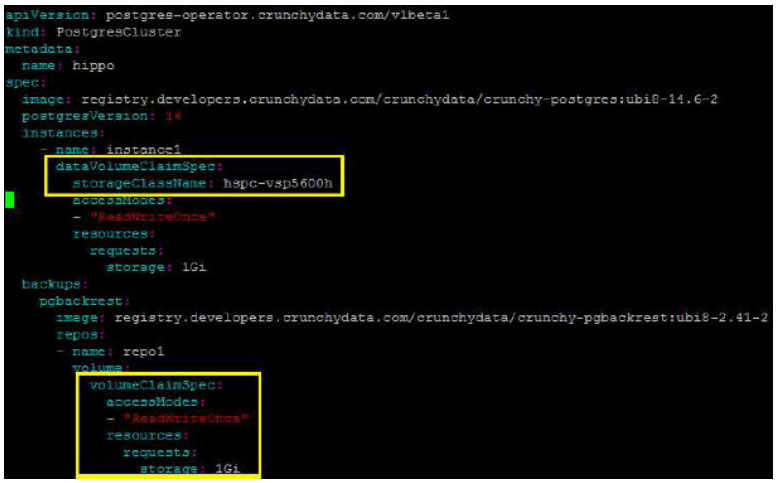
You can select or enter more details, based on application requirements. See https://argo-cd.readthedocs.io/en/stable/getting_started/#6-create-an-application-from-a-git-repository for detailed instructions.

Deploy the Crunchy Postgres Operator

Complete this procedure to deploy Crunchy Postgres Operator.

1. Update the Postgres definition file.

   To create PVs and store Postgres data on Hitachi Virtual Storage Platform (VSP), the Crunchy Postgres Operator definition file (path: kustomize/postgres) must be updated and a Storage Class must be added to that file. By adding this storageClass, Storage Plug-in for Containers will be triggered at the time of deployment, and a persistent volume with attributes that are configured on the same definition file will be created.



2. Create a Crunchy Postgres Operator.

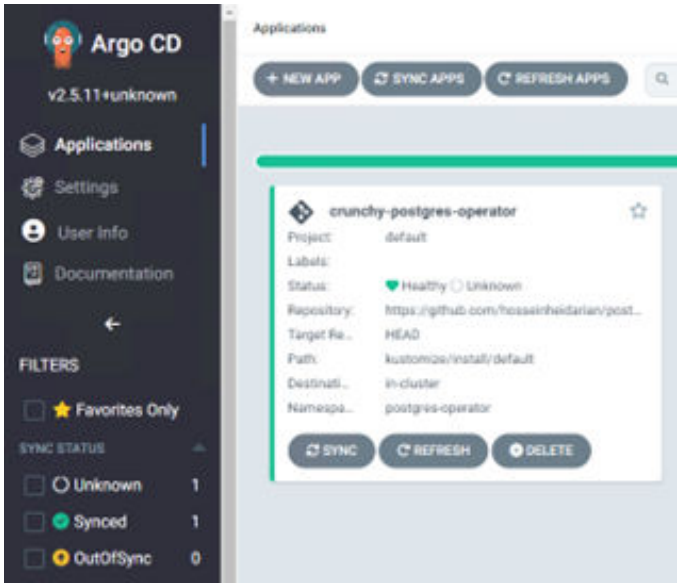
   At this point you need to create a namespace. You can do it manually by running the following command, or update the definition file by adding a namespace, or a better option is to let Argo CD create the namespace for you. In this example Argo CD created the namespace postgres-operator.

   ```
   oc create namespace postgres-operator
   ```

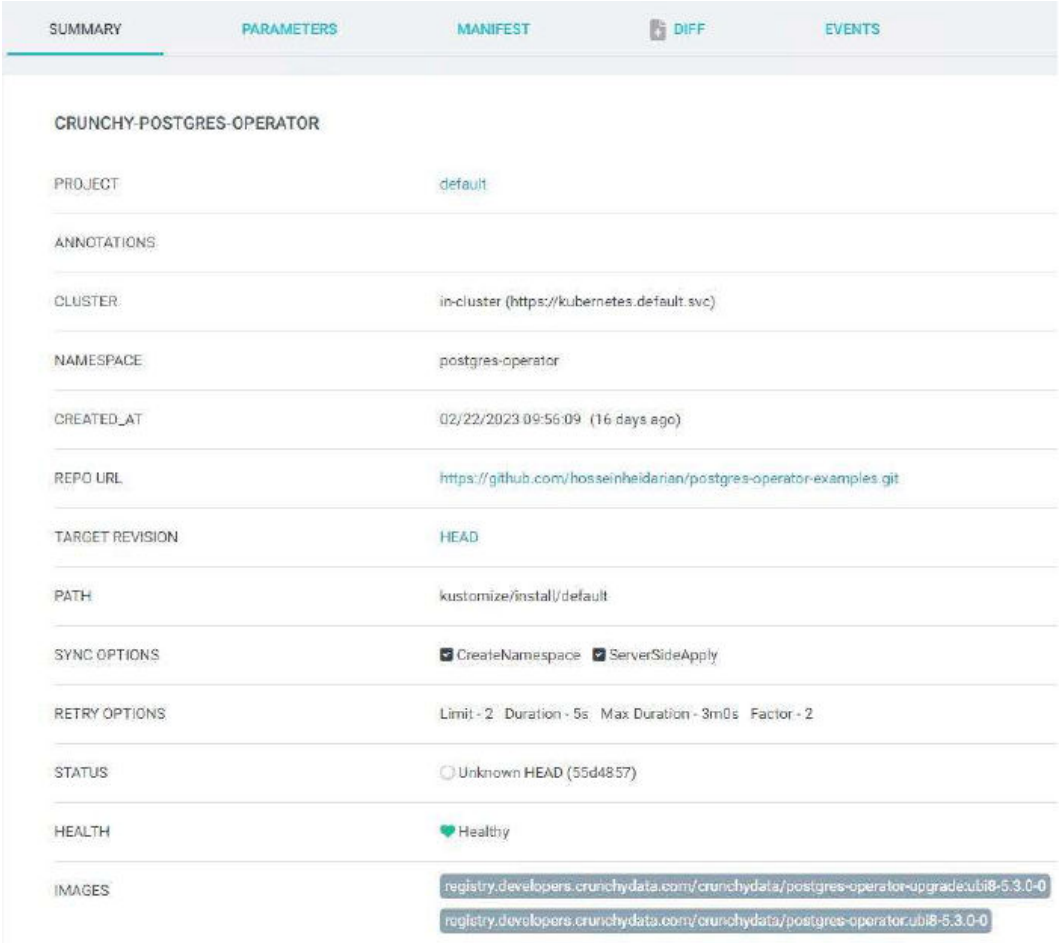
   To create the Crunchy application, follow the steps in the Application deployment section. The following are the inputs that need to be entered on the create Crunchy

Postgres Operator application creation page:

- General
  - Application Name: crunchy-postgres-operator
  - Project: default
  - SYNC POLICY: Automatic
    - PRUNE RESOURCES: Checked
    - SELF HEAL: Checked
    - SET DELETION FINALIZER: Checked
  - SYNC OPTION
    - AUTO-CREATED NAMESPACE
    - SERVER-SIDE APPLY
  - PRUNE PROPAGATION: foreground
  - RETRY: Checked
    - Limit: 2 Duration:5s
    - Max Duration: 3m0s Factor:2
- Source
  - Repository: https://github.com/hosseinheidarian/postgres-operator-example.git (you will need a GitHub login)
  - Path: kustomize/install/default
- Destination
  - Cluster URL: https://kubernetes.default.svc
  - Namespace: postgres-operator



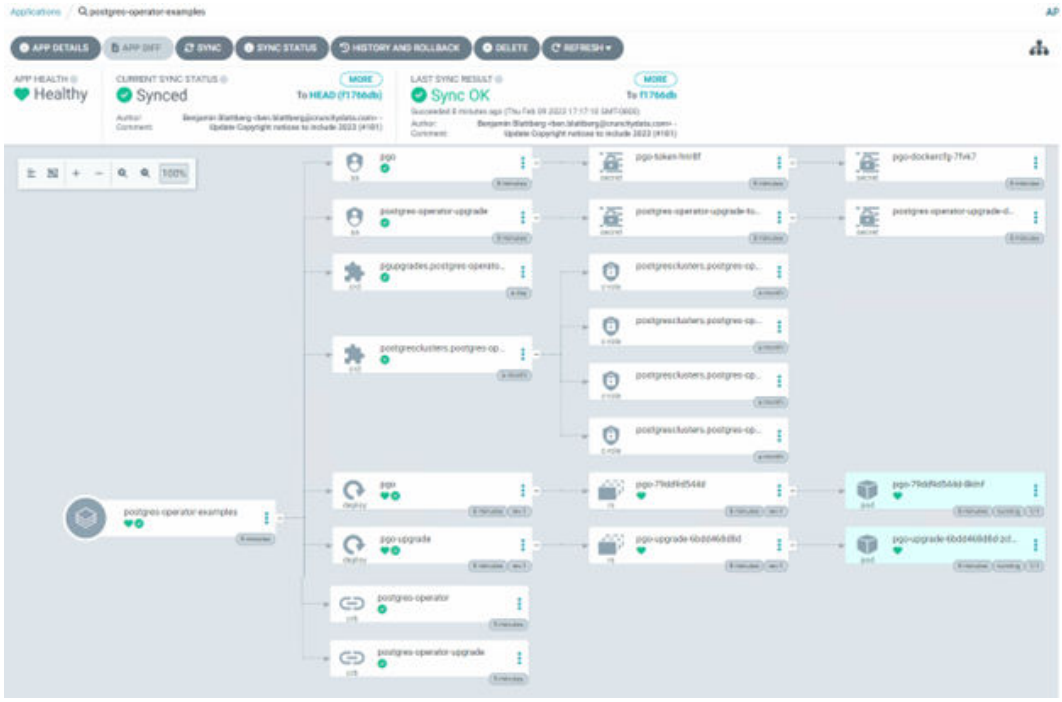
The following is an example Crunchy-postgres-Operator application SUMMARY.



The following is an example Crunchy-postgres-Operator application MANIFEST file.

```
project: default
source:
  repoURL: 'https://github.com/hosseinheidarian/postgres-operator-examples.git'
  path: kustomize/install/default
  targetRevision: HEAD
destination:
  server: 'https://kubernetes.default.svc'
  namespace: postgres-operator
syncPolicy:
  automated:
    prune: true
    selfHeal: true
  syncOptions:
    - CreateNamespace=true
    - ServerSideApply=true
  retry:
    limit: 2
    backoff:
      duration: 5s
      factor: 2
      maxDuration: 3m0s

```

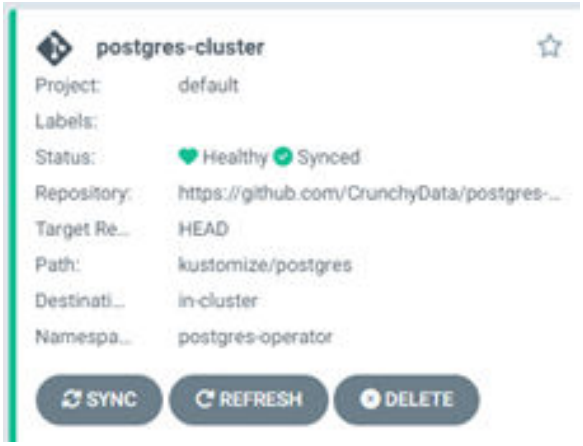


3. Create a Crunchy Postgres cluster.

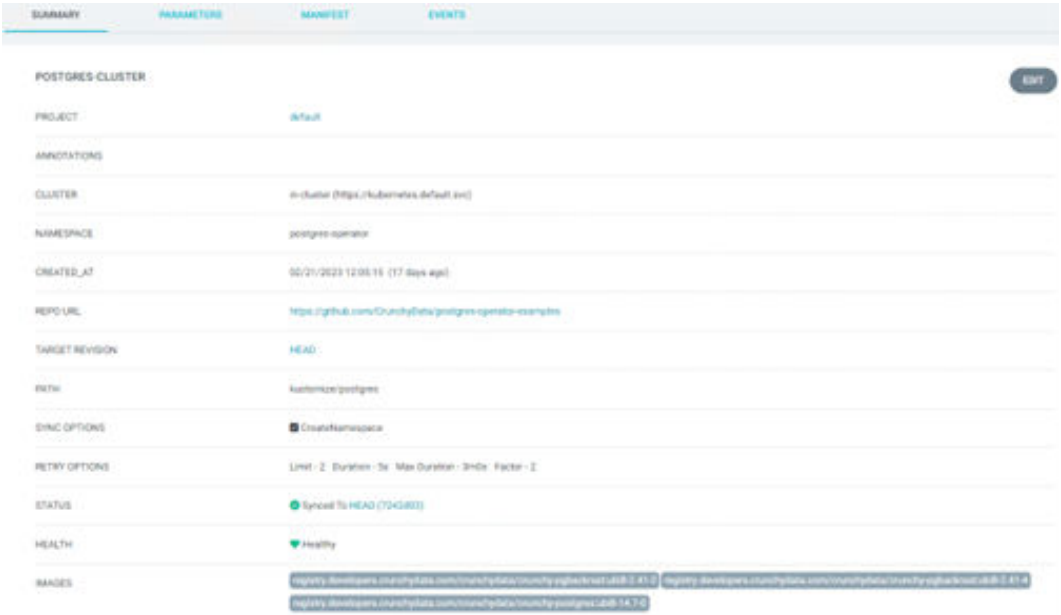
   To create Crunchy applications, follow all the steps in the application deployment section. The following are the inputs that are entered on the Create Crunchy Postgres

Operator application creation page:

- General
  - Application Name: postgres-cluster
  - Project: default
  - SYNC POLICY: Automatic
    - PRUNE RESOURCES: Checked
    - SELF HEAL: Checked
    - SET DELETION FINALIZER: Checked
  - SYNC OPTION
    - AUTO-CREATED NAMESPACE
    - SERVER-SIDE APPLY
  - PRUNE PROPAGATION: foreground
  - RETRY: Checked
    - Limit: 2 Duration:5s
    - Max Duration: 3m0s Factor:2
- Source
  - Repository: https://github.com/hosseinheidarian/postgres-operator-example.git (you will need a GitHub login)
  - Path: kustomize/postgres
- Destination
  - Cluster URL: https://kubernetes.default.svc
  - Namespace: postgres-operator



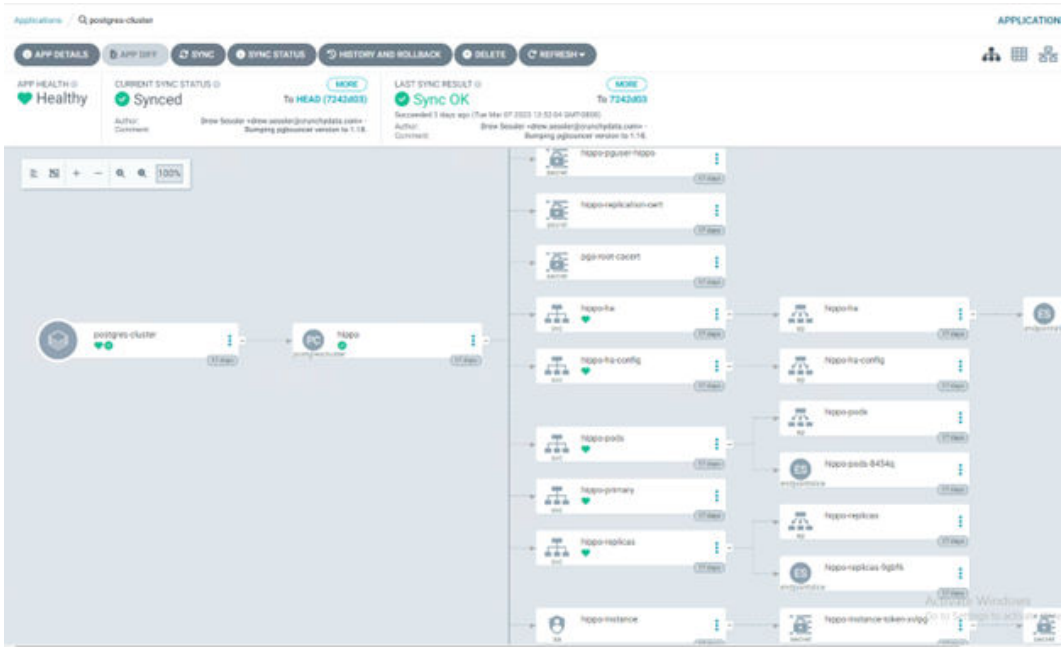
The following is an example postgres-cluster SUMMARY.



The following is an example postgres-cluster MANIFEST file.

SUMMARY PARAMETERS MANIFEST EVENTS

```
project: default
source:
  repoURL: 'https://github.com/CrunchyData/postgres-operator-examples'
  path: kustomize/postgres
  targetRevision: HEAD
destination:
  server: 'https://kubernetes.default.svc'
  namespace: postgres-operator
syncPolicy:
  automated:
    prune: true
    selfHeal: true
  syncOptions:
    - CreateNamespace=true
  retry:
    limit: 2
    backoff:
      duration: 5s
      factor: 2
      maxDuration: 3m0s
```



4. Validate the deployment.

   To verify the Crunchy Postgres Operator and Postgres Cluster deployment, do the following.

   The following command shows you the namespace postgres-operator that was created by Argo CD.

```
oc get namespace | grep postgres-operator
```

```
[ocpinstall@hh02-ocp-admin-ws]$oc get namespaces | grep postgres-operator
postgres-operator                                 Active   17d
[ocpinstall@hh02-ocp-admin-ws]$
```

The following command sets the active working namespace to postgres-operator.

```
oc project postgres-operator
```

```
[ocpinstall@hh02-ocp-admin-ws]$oc get namespaces | grep postgres-operator
postgres-operator                                 Active   17d
[ocpinstall@hh02-ocp-admin-ws]$
```

View all created pods.

```
oc get pods
```

```
[ocpinstall@hh02-ocp-admin-ws]$oc get pods
NAME                                READY   STATUS      RESTARTS   AGE
hippo-backup-c9bz-qclxh             0/1     Completed   0          17d
hippo-instance1-bt4m-0              4/4     Running     0          2d23h
hippo-repo-host-0                   2/2     Running     0          2d23h
pgo-54c4d68479-g9lzg                1/1     Running     1          16d
pgo-upgrade-6d74d4b8cf-xw84s        1/1     Running     1          16d
[ocpinstall@hh02-ocp-admin-ws]$
```

The following command shows all the available services on namespace postgres-operator.

```
oc get svc
```

```
[ocpinstall@hh02-ocp-admin-ws]$oc get svc
NAME                TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
hippo-ha            ClusterIP   172.30.208.155   <none>        5432/TCP   17d
hippo-ha-config     ClusterIP   None             <none>        <none>     17d
hippo-pods          ClusterIP   None             <none>        <none>     17d
hippo-primary       ClusterIP   None             <none>        5432/TCP   17d
hippo-replicas      ClusterIP   172.30.44.170    <none>        5432/TCP   17d
[ocpinstall@hh02-ocp-admin-ws]$
```

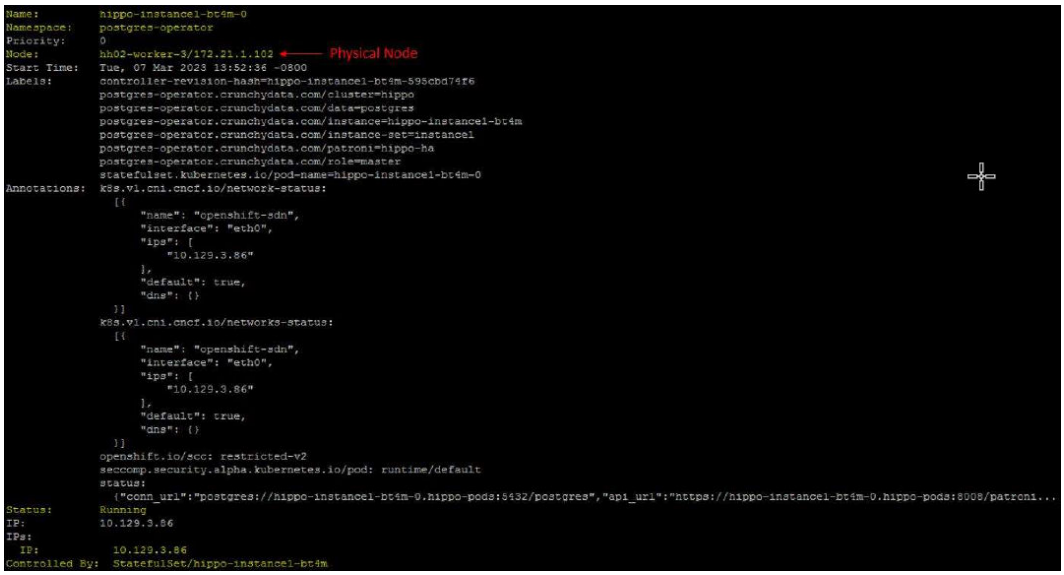
List created PVCs on Hitachi VSP by running the following command.

```
oc get pvc
```

```
[ocpinstall@hh02-ocp-admin-ws]$oc get pvc
NAME                          STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS    AGE
hippo-instance1-bt4m-pgdata   Bound    pvc-c4e4e5de-177a-42e4-86d9-530754a53a0b   1Gi        RWO            hspc-vsp5600h   17d
hippo-repo1                   Bound    pvc-7398c1db-69ff-4d2c-ae57-146f3399d8c0   1Gi        RWO            hspc-vsp5600h   17d
[ocpinstall@hh02-ocp-admin-ws]$
```

View hippo instance details.

```
oc describe pod -n hippo-instance1-bt4m-0
```



We have now successfully deployed a Postgres database using GitOps in a beginner-friendly manner. With the help of Storage Plug-in for Containers and VSP, persistent volumes used by the Postgres database were automatically allocated, eliminating the need for manual intervention. As a result, we have effectively demonstrated a simple GitOps-driven framework for deploying modern databases on the Hitachi Unified Compute Platform (UCP) Kubernetes Solution with Red Hat OpenShift.

Conclusion

In summary, this paper demonstrates an example project of the deployment and operation of a sample configuration on top of an OpenShift Cluster (version 4.12) with three virtual master nodes and two virtual and one physical worker node, including the physical worker node connected to Hitachi Virtual Storage Platform (VSP) using an Emulex HBA on a Fibre Channel network.

The GitOps-driven framework presented in this paper offers a clear and concise methodology for deploying and managing containerized data applications (specifically PostgreSQL in this paper) on OpenShift clusters on UCP with Storage Plug-in for Containers and VSP.

References

- Hitachi Storage Plug-in for Containers

  https://knowledge.hitachivantara.com/Documents/Adapters_and_Drivers/Storage_Adapters_and_Drivers/Containers/Storage_Plug-in_for_Containers

- Argo CD Getting Started Guide

  https://argo-cd.readthedocs.io/en/stable/getting_started/

- Crunchy Postgres Operator documentation

  https://access.crunchydata.com/documentation/postgres-operator/v5/

Hitachi Vantara

Corporate Headquarters
2535 Augustine Drive
Santa Clara, CA 95054 USA

HitachiVantara.com/contact