AWS Open Source Blog

Continuous Delivery using Spinnaker on Amazon EKS

by Irshad Buchh | on 06 NOV 2019 | in Amazon Elastic Kubernetes Service, AWS Cloud9, Open Source | Permalink | Comments | Share

I work closely with partners, helping them to architect solutions on AWS for their customers. Customers running their microservices-based applications on Amazon Elastic Kubernetes Service (Amazon EKS) are looking for guidance on architecting complete end-to-end Continuous Integration (CI) and Continuous Deployment/Delivery (CD) pipelines using Jenkins and Spinnaker. The benefits of using Jenkins include that it is a very popular CI server with great community support and it has many plugins (Slack, GitHub, Docker, Build Pipeline) available. Spinnaker provides automated release, built-in deployment, and supports blue/green deployment out of the box. This post focuses on Continuous Delivery, and will discuss the installation and configuration of Spinnaker on Amazon EKS.

You can also refer to an earlier post, *Build a Deployment Pipeline with Spinnaker on Kubernetes,* in which Prabhat Sharma explained some of the fundamental concepts of Spinnaker.

Overview of concepts

Amazon EKS runs the Kubernetes management infrastructure across multiple AWS Availability Zones, automatically detects and replaces unhealthy control plane nodes, and provides on-demand upgrades and patching. You simply provision worker nodes and connect them to the provided Amazon EKS endpoint.

In software engineering, **continuous integration** is the practice of merging all developers' working copies to a shared mainline several times a day. Grady Booch first proposed the term CI in 1991, though he did not advocate integrating several times a day.

Continuous delivery is a software engineering approach in which teams produce software in short cycles, ensuring that the software can be reliably released at any time and, when releasing the software, doing so manually. This approach aims at building, testing, and releasing software with greater speed and frequency.

Continuous deployment is a strategy for software releases wherein any code commit that passes the **automated** testing phase is automatically released into the production environment, making changes that are visible to the software's users.

Prerequisites

In order to implement the instructions laid out in this post, you will need the following:

- An AWS account
- A Docker Hub account
- A GitHub account

Architecture

In this post, I will discuss the following architecture for Continuous Delivery:

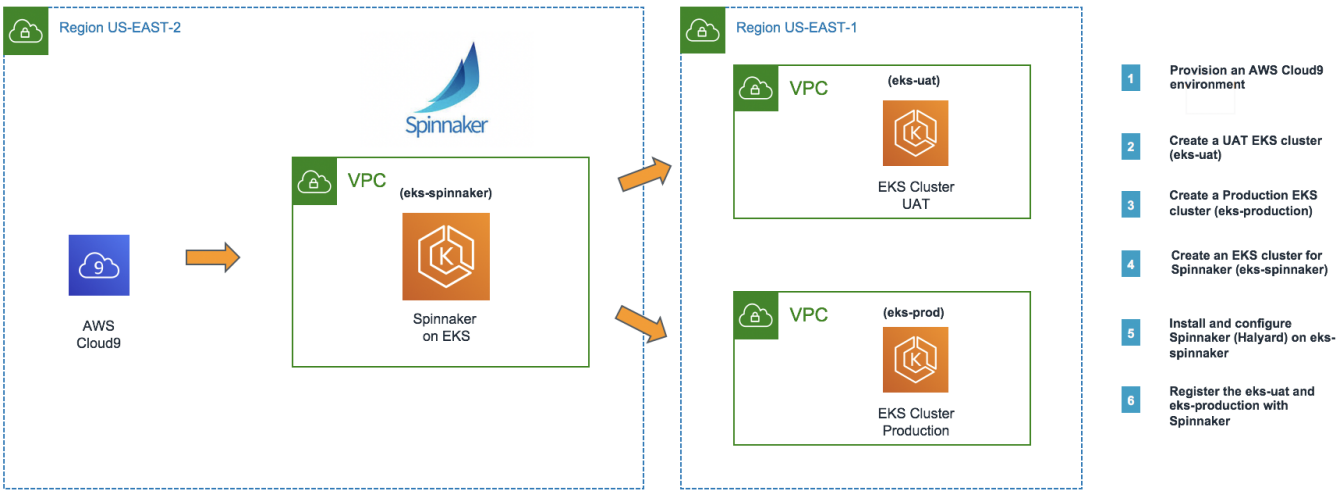


Fig 1. Continuous Delivery Architecture

Here are the steps we'll be following to create the continuous delivery architecture:

- Create an AWS Cloud9 environment
- Configure AWS Cloud9 environment
- Create Amazon EKS clusters
- Install and configure Spinnaker
- Cleanup

Create an AWS Cloud9 environment

Log into the AWS Management Console and search for Cloud9 services in the search bar:

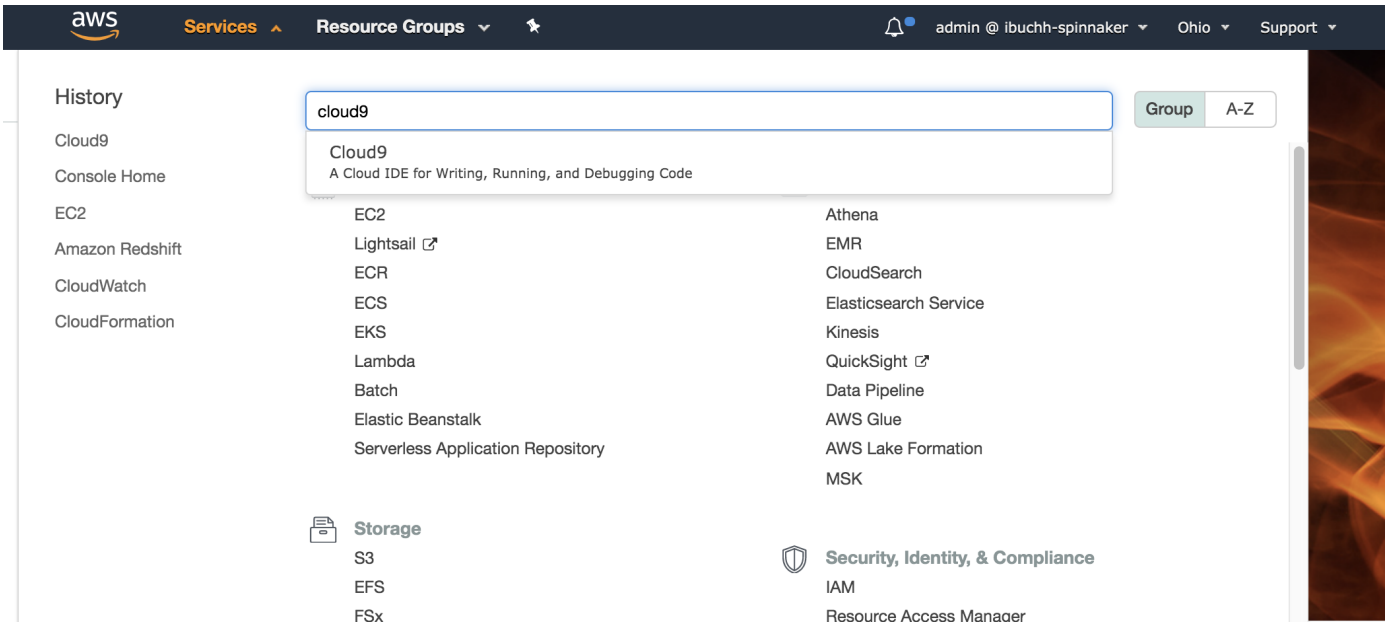


Fig 2. AWS Management Console

Click **Cloud9** and create an AWS Cloud9 environment in the us-east-2 region based on Ubuntu Server 18.04 LTS (Halyard is not supported on Amazon Linux yet). Choose the settings as shown in Fig 3 where the platform should be Ubuntu Server 18.04 LTS.

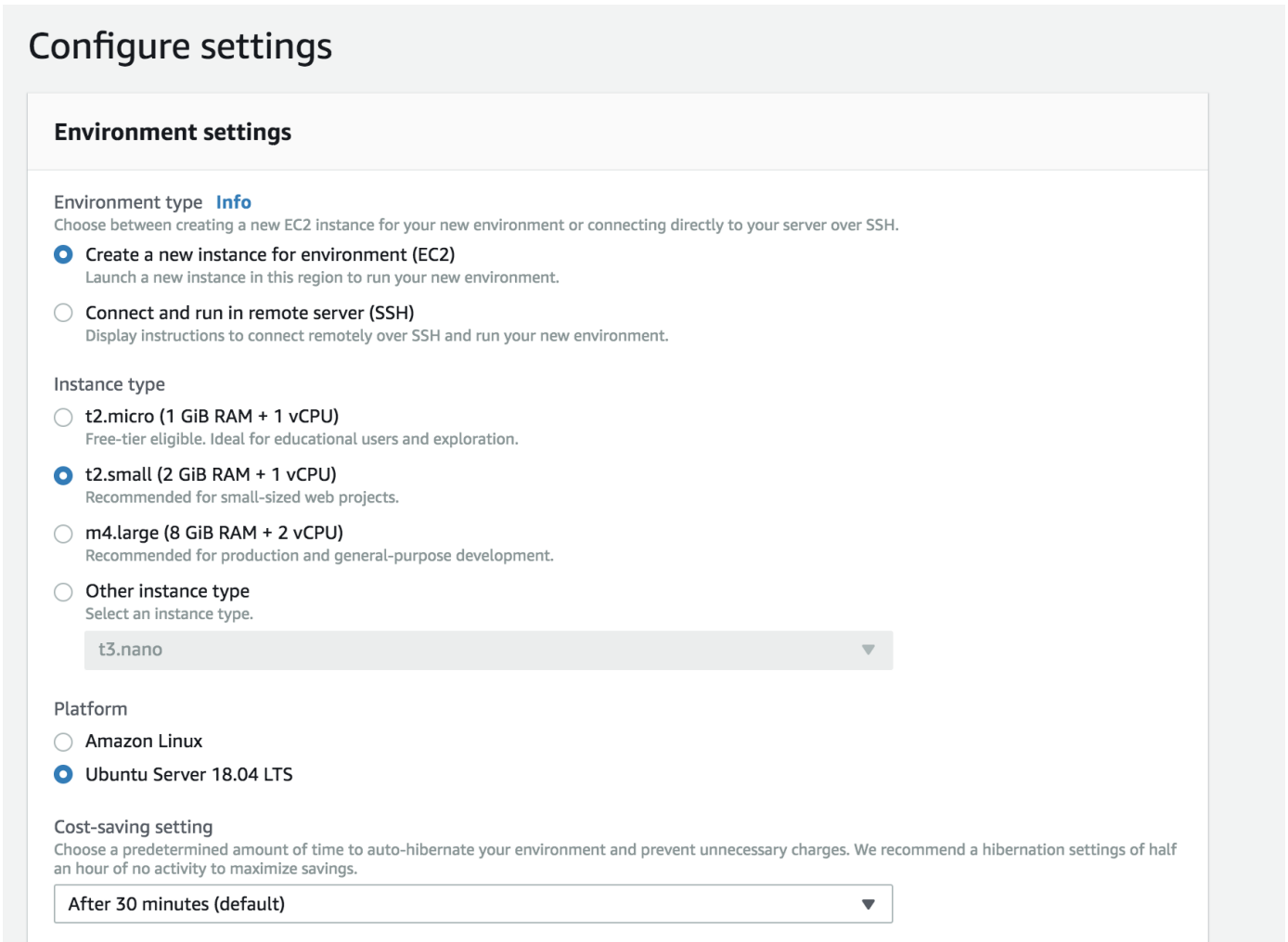


Fig 3. AWS Cloud9 settings

Configure the AWS Cloud9 environment

Launch the AWS Cloud9 IDE. In a new terminal session, follow the instructions to configure the AWS Cloud9 environment.

1. Install and configure Kubectl

Install `kubectl` and `aws-iam-authenticator` on the AWS Cloud9 Ubuntu machine:

```
Bash
 curl -LO https://storage.googleapis.com/kubernetes-release/release/$(curl -s htt

 chmod +x ./kubectl

 sudo mv ./kubectl /usr/local/bin/kubectl

 curl -o aws-iam-authenticator https://amazon-eks.s3-us-west-2.amazonaws.com/1.13

 chmod +x ./aws-iam-authenticator

 mkdir -p $HOME/bin && cp ./aws-iam-authenticator $HOME/bin/aws-iam-authenticator

 echo 'export PATH=$HOME/bin:$PATH' >> ~/.bashrc

 aws-iam-authenticator  help
```

The script verifies that `aws-iam-authenticator` is working by displaying the help contents of `aws-iam-authenticator`.

2. Upgrade awscli

```
Bash
 aws --version
 pip install awscli --upgrade --user
```

3. Install eksctl

```
Bash
 curl --silent --location "https://github.com/weaveworks/eksctl/releases/download
```

```
Bash
 sudo mv /tmp/eksctl /usr/local/bin
```

4. Install Terraform

```
Bash
wget https://releases.hashicorp.com/terraform/0.12.4/terraform_0.12.4_linux_amd6
unzip terraform_0.12.4_linux_amd64.zip
sudo mv terraform /usr/local/bin/
export PATH=$PATH:/usr/local/bin/terraform
```

5. Install Halyard

```
Bash
curl -O https://raw.githubusercontent.com/spinnaker/halyard/master/install/debia
sudo bash InstallHalyard.sh
sudo update-halyard
hal -v
```

Create Amazon EKS clusters

To make a complete environment, I will create three AWS EKS clusters including one for production, one for UAT, and one for Spinnaker installation. Inside the AWS Cloud9 IDE, run the following commands to create these Amazon EKS clusters. (You can choose your preferred regions; for this post I shall use us-east-2 to provision the Amazon EKS cluster for Spinnaker deployment and us-east-1 region to provision the UAT and production Amazon EKS clusters.

1. Create the Production Amazon EKS cluster

```
Bash
eksctl create cluster --name=eks-prod --nodes=3 --region=us-east-1 \
  --write-kubeconfig=false
```

```
$
$eksctl create cluster --name=eks-prod --nodes=3 --region=us-east-1 --write-kubeconfig=false

[ℹ]  eksctl version 0.7.0
[ℹ]  using region us-east-1
[ℹ]  setting availability zones to [us-east-1c us-east-1a]
[ℹ]  subnets for us-east-1c - public:192.168.0.0/19 private:192.168.64.0/19
[ℹ]  subnets for us-east-1a - public:192.168.32.0/19 private:192.168.96.0/19
[ℹ]  nodegroup "ng-ca6f0faf" will use "ami-0392bafc801b7520f" [AmazonLinux2/1.14]
[ℹ]  using Kubernetes version 1.14
[ℹ]  creating EKS cluster "eks-prod" in "us-east-1" region
[ℹ]  will create 2 separate CloudFormation stacks for cluster itself and the initial nodegroup
[ℹ]  if you encounter any issues, check CloudFormation console or try 'eksctl utils describe-stacks --region=us-east-1 --name=eks-prod'
[ℹ]  CloudWatch logging will not be enabled for cluster "eks-prod" in "us-east-1"
[ℹ]  you can enable it with 'eksctl utils update-cluster-logging --region=us-east-1 --name=eks-prod'
[ℹ]  2 sequential tasks: { create cluster control plane "eks-prod", create nodegroup "ng-ca6f0faf" }
[ℹ]  building cluster stack "eksctl-eks-prod-cluster"
[ℹ]  deploying stack "eksctl-eks-prod-cluster"
[ℹ]  building nodegroup stack "eksctl-eks-prod-nodegroup-ng-ca6f0faf"
[ℹ]  --nodes-min=3 was set automatically for nodegroup ng-ca6f0faf
[ℹ]  --nodes-max=3 was set automatically for nodegroup ng-ca6f0faf
[ℹ]  deploying stack "eksctl-eks-prod-nodegroup-ng-ca6f0faf"
[✔]  all EKS cluster resources for "eks-prod" have been created
[ℹ]  adding identity "arn:aws:iam::268453042196:role/eksctl-eks-prod-nodegroup-ng-ca6f-NodeInstanceRole-1QQ8OUHYBZYVZ" to auth ConfigMap
[ℹ]  nodegroup "ng-ca6f0faf" has 0 node(s)
[ℹ]  waiting for at least 3 node(s) to become ready in "ng-ca6f0faf"
[ℹ]  nodegroup "ng-ca6f0faf" has 3 node(s)
[ℹ]  node "ip-192-168-34-66.ec2.internal" is ready
[ℹ]  node "ip-192-168-5-115.ec2.internal" is ready
[ℹ]  node "ip-192-168-9-138.ec2.internal" is ready
```

Fig 4. eksctl

2. Create the UAT Amazon EKS cluster

```
Bash
 eksctl create cluster --name=eks-uat --nodes=3 --region=us-east-1 \
   --write-kubeconfig=false
```

3. Create the Spinnaker Amazon EKS cluster

```
Bash
 eksctl create cluster --name=eks-spinnaker --nodes=2 --region=us-east-2 \
  --write-kubeconfig=false
```

eksctl is a simple CLI tool for creating clusters on Amazon EKS which creates the following components of the Amazon EKS cluster architecture:

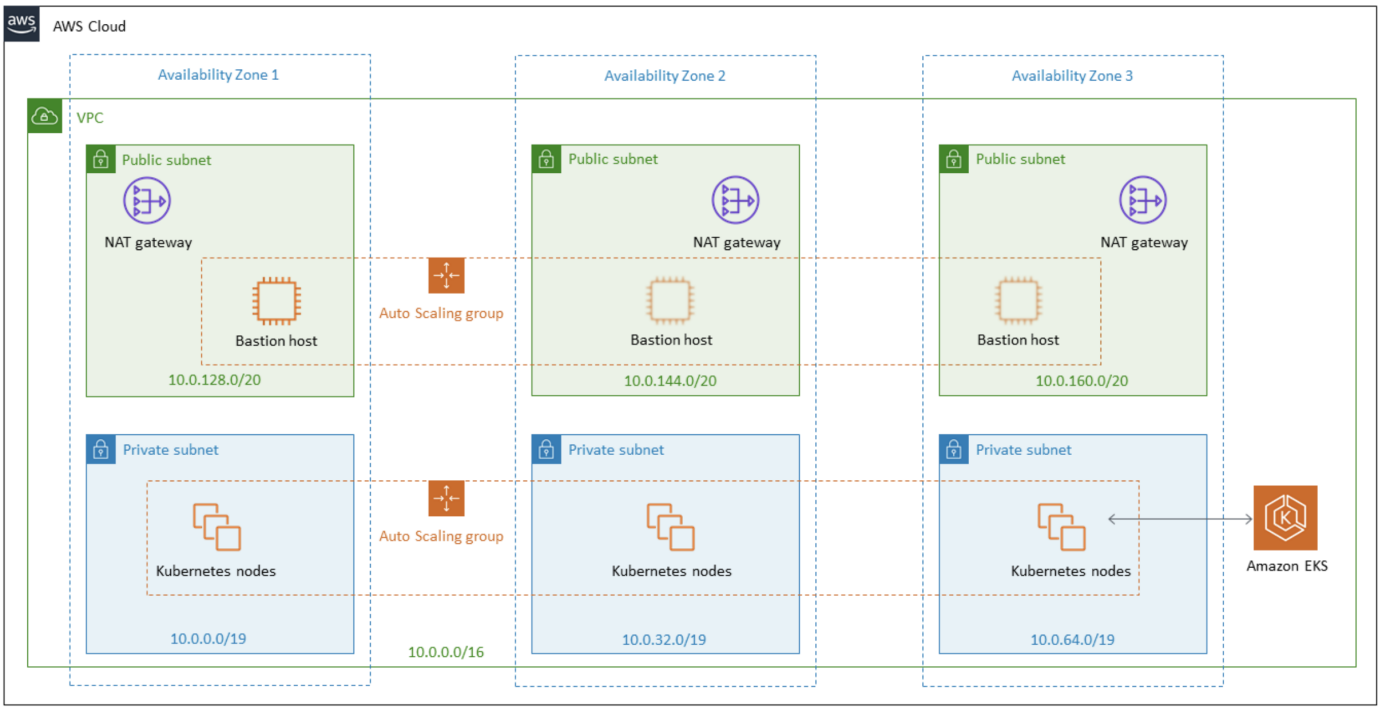


Fig 5. Amazon EKS cluster

Install and configure Spinnaker

This section will walk you through the process of installing and configuring Spinnaker for use with Amazon EKS. I prefer to use Armory Spinnaker because:

- Armory provides an installer that does many of the configurations required with a command `hal armory init.` This configuration supports AWS Simple Storage Service S3.
- Armory provides pipelines as code so that you can store pipeline configurations in source control and have a consistent, versioned method of application deployment. In the op you can only create pipelines through the UI.
- Armory develops native integrations of Spinnaker with third party tools (https://www.armory.io/integrations).

1. Retrieve Amazon EKS cluster kubectl contexts

```
Bash
 aws eks update-kubeconfig --name eks-spinnaker --region us-east-2 \
  --alias eks-spinnaker

 aws eks update-kubeconfig --name eks-uat --region us-east-1 \
  --alias eks-uat

 aws eks update-kubeconfig --name eks-prod --region us-east-1 \
  --alias eks-prod
```

2. Check halyard version

```
Bash
 hal -v
```

3. Create and configure a Docker registry

```
Bash
 hal config provider docker-registry enable

 hal config provider docker-registry account add ibuchh-docker \
  --address index.docker.io --username ibuchh --password
```

This command will prompt you to enter your docker account password.

4. Add and configure a GitHub account

```
Bash
 hal config artifact github enable

 hal config artifact github account add spinnaker-github --username ibuchh \
   --password --token
```

This command will prompt you to enter your GitHub token that you can get from the GitHub account setting.

5. Add and configure Kubernetes accounts

Production Amazon EKS account:

Set the Kubernetes provider as enabled:

```
Bash
 hal config provider kubernetes enable

 kubectl config use-context eks-prod
```

A context element in a kubeconfig file is used to group access parameters under a convenient name. Each context has three parameters: cluster, namespace, and user. By default, the kubectl command line tool uses parameters from the current context to communicate with the cluster.

```
Bash
 CONTEXT=$(kubectl config current-context)
```

We will create service accounts for the three Amazon EKS clusters. See the Kubernetes documentation for more details on service accounts.

```
Bash
 kubectl apply --context $CONTEXT \
     -f https://spinnaker.io/downloads/kubernetes/service-account.yml
```

Extract the secret token of the `spinnaker-service-account` :

```
Bash
 TOKEN=$(kubectl get secret --context $CONTEXT \
    $(kubectl get serviceaccount spinnaker-service-account \
        --context $CONTEXT \
        -n spinnaker \
        -o jsonpath='{.secrets[0].name}') \
    -n spinnaker \
    -o jsonpath='{.data.token}' | base64 --decode)
```

Set the user entry in `kubeconfig:`

```
Bash
 kubectl config set-credentials ${CONTEXT}-token-user --token $TOKEN

 kubectl config set-context $CONTEXT --user ${CONTEXT}-token-user
```

Add `eks-prod cluster` as a Kubernetes provider.

```
Bash
 hal config provider kubernetes account add eks-prod --provider-version v2 \
  --docker-registries ibuchh-docker --context $CONTEXT
```

UAT Amazon EKS account:

```
Bash
 kubectl config use-context eks-uat

 CONTEXT=$(kubectl config current-context)

 kubectl apply --context $CONTEXT \
     -f https://spinnaker.io/downloads/kubernetes/service-account.yml
```

Extract the secret token of the `spinnaker-service-account` :

```
Bash
 TOKEN=$(kubectl get secret --context $CONTEXT \
    $(kubectl get serviceaccount spinnaker-service-account \
        --context $CONTEXT \
        -n spinnaker \
        -o jsonpath='{.secrets[0].name}') \
    -n spinnaker \
    -o jsonpath='{.data.token}' | base64 --decode)
```

Set the service account entry in `kubeconfig` file:

```
Bash
 kubectl config set-credentials ${CONTEXT}-token-user --token $TOKEN

 kubectl config set-context $CONTEXT --user ${CONTEXT}-token-user
```

Add `eks-uat` cluster as a Kubernetes provider.

```
Bash
 hal config provider kubernetes account add eks-uat --provider-version v2 \
  --docker-registries ibuchh-docker --context $CONTEXT
```

Spinnaker Amazon EKS account:

```
Bash
 kubectl config use-context eks-spinnaker

 CONTEXT=$(kubectl config current-context)

 kubectl apply --context $CONTEXT \
     -f https://spinnaker.io/downloads/kubernetes/service-account.yml
```

Extract the secret token of the `spinnaker-service-account` :

```
Bash
 TOKEN=$(kubectl get secret --context $CONTEXT \
    $(kubectl get serviceaccount spinnaker-service-account \
        --context $CONTEXT \
        -n spinnaker \
        -o jsonpath='{.secrets[0].name}') \
    -n spinnaker \
    -o jsonpath='{.data.token}' | base64 —decode)
```

Set the service account entry in the Kubeconfig file:

```
Bash
 kubectl config set-credentials ${CONTEXT}-token-user --token $TOKEN

 kubectl config set-context $CONTEXT --user ${CONTEXT}-token-user
```

Add `eks-spinnaker` cluster as a Kubernetes provider.

```
Bash
 hal config provider kubernetes account add eks-spinnaker --provider-version v2 \
  --docker-registries ibuchh-docker  --context $CONTEXT
```

6. Enable artifact support

```
Bash
 hal config features edit --artifacts true
```

7. Configure Spinnaker to install in Kubernetes

For our environment we will use a distributed Spinnaker installation onto the Kubernetes cluster. This installation model has Halyard deploy each of the Spinnaker microservices separately. A distributed installation helps to limit update-related downtime, making it recommended for use in production environments.

```
Bash
 hal config deploy edit --type distributed --account-name eks-spinnaker
```

8. Configure Spinnaker to use AWS S3

You will need your AWS account access key and secret access key.

```
Bash
 export YOUR_ACCESS_KEY_ID=<access-key>

 hal config storage s3 edit --access-key-id $YOUR_ACCESS_KEY_ID \
  --secret-access-key --region us-east-2
```

Enter your AWS account secret access key at the prompt.

```
Bash
 hal config storage edit --type s3
```

9. Choose the Spinnaker version

To identify the latest version of Spinnaker to install, run the following to get a list of available versions:

```
Bash
 hal version list
```

At the time of writing, 1.15.0 is the latest Spinnaker version:

```
Bash
 export VERSION=1.15.0

 hal config version edit --version $VERSION
```

Now we are finally ready to install Spinnaker on the eks-spinnaker Amazon EKS cluster.

```
Bash
 hal deploy apply
```

10. Verify the Spinnaker installation

```
Bash
 kubectl -n spinnaker get svc
```

```
Administrator:~/environment $ kubectl get svc -n spinnaker
NAME                TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
spin-clouddriver    ClusterIP   10.100.226.113   <none>        7002/TCP   7m11s
spin-deck           ClusterIP   10.100.178.237   <none>        9000/TCP   7m13s
spin-echo           ClusterIP   10.100.239.74    <none>        8089/TCP   7m12s
spin-front50        ClusterIP   10.100.153.60    <none>        8080/TCP   7m15s
spin-gate           ClusterIP   10.100.72.90     <none>        8084/TCP   7m14s
spin-igor           ClusterIP   10.100.149.98    <none>        8088/TCP   7m13s
spin-orca           ClusterIP   10.100.9.100     <none>        8083/TCP   7m12s
spin-redis          ClusterIP   10.100.217.9     <none>        6379/TCP   7m10s
spin-rosco          ClusterIP   10.100.0.212     <none>        8087/TCP   7m13s
```

Fig 6. Spinnaker Microservices

11. Expose Spinnaker using Elastic Loadbalancer

I shall expose the Spinnaker API (Gate) and the Spinnaker UI (Deck) via Load Balancers by running the following commands to create the `spin-gate-public` and `spin-deck-public services` :

```
Bash
 export NAMESPACE=spinnaker

 kubectl -n ${NAMESPACE} expose service spin-gate --type LoadBalancer \
   --port 80 --target-port 8084 --name spin-gate-public

 kubectl -n ${NAMESPACE} expose service spin-deck --type LoadBalancer \
   --port 80 --target-port 9000 --name spin-deck-public

 export API_URL=$(kubectl -n $NAMESPACE get svc spin-gate-public \
  -o jsonpath='{.status.loadBalancer.ingress[0].hostname}')

 export UI_URL=$(kubectl -n $NAMESPACE get svc spin-deck-public  \
  -o jsonpath='{.status.loadBalancer.ingress[0].hostname}')

 hal config security api edit --override-base-url http://${API_URL}

 hal config security ui edit --override-base-url http://${UI_URL}

 hal deploy apply
```

12. Re-verify the Spinnaker installation

```
Bash
 kubectl -n spinnaker get svc
```

```
$
$kubectl get svc -n spinnaker
NAME              TYPE           CLUSTER-IP       EXTERNAL-IP
spin-clouddriver  ClusterIP      10.100.224.117   <none>
spin-deck         ClusterIP      10.100.20.41     <none>
spin-deck-public  LoadBalancer   10.100.154.221   [redacted].us-east-2.elb.amazonaws.com
spin-echo         ClusterIP      10.100.97.220    <none>
spin-front50      ClusterIP      10.100.185.200   <none>
spin-gate         ClusterIP      10.100.205.117   <none>
spin-gate-public  LoadBalancer   10.100.240.190   [redacted].us-east-2.elb.amazonaws.com
spin-igor         ClusterIP      10.100.242.203   <none>
spin-orca         ClusterIP      10.100.80.2      <none>
spin-redis        ClusterIP      10.100.190.200   <none>
spin-rosco        ClusterIP      10.100.88.67     <none>
$
```

Fig 7. Spinnaker UI endpoints

13. Log in to Spinnaker console

Using a browser, log in to the Spinnaker UI using the `spin-deck-public services` endpoint as shown above.

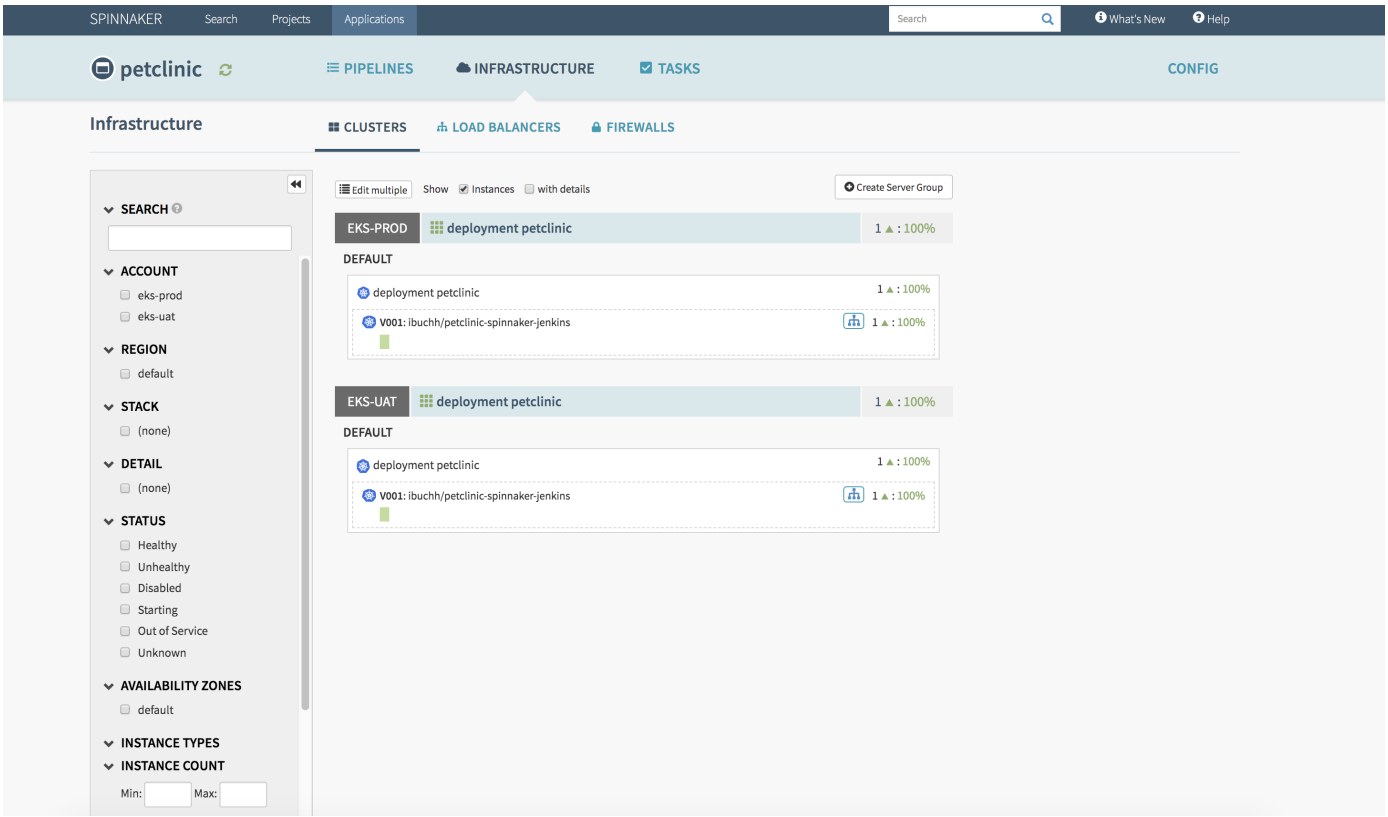


Fig 8. Spinnaker Console

Cleanup

To remove the three Amazon EKS clusters, run the following commands inside the AWS Cloud9 IDE:

```
Bash
 eksctl delete cluster --name=eks-uat --region=us-east-1

 eksctl delete cluster --name=eks-prod --region=us-east-1

 eksctl delete cluster --name=eks-spinnaker --region=us-east-2
```

Conclusion

In this post, I have outlined the detailed instructions needed to install and configure a Continuous Delivery platform using Spinnaker (Halyard) on Amazon EKS. Spinnaker can integrate with Jenkins to architect complete Continuous Integration (CI) and Continuous Deployment/Delivery (CD) pipelines. To learn more, see the Spinnaker documentation.

If you have questions or suggestions, please comment below.

re:Invent 2019!

Irshad will give session OPN309 – CI/CD using Jenkins and Spinnaker on Amazon EKS – reserve your seat now!

TAGS: Amazon EKS, AWS Cloud9, CI/CD, DevOps, Halyard, IDE, Jenkins, kubernetes, Spinnaker

AWS Partner Network

Find an APN member to support your cloud business needs

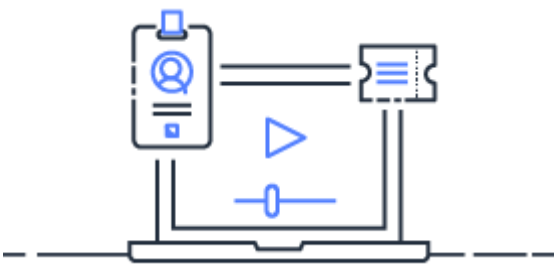
Learn more »

Resources

Open Source at AWS
Projects on GitHub

Follow

AWS Open Source
AWS Cloud
Facebook
LinkedIn
Twitch
Open Source RSS
Email Updates



AWS Events

Discover the latest AWS events in your region

Learn more »

Related Posts

Kubernetes Logging powered by AWS for Fluent Bit

Monitoring Kubernetes Environments with AWS and New Relic's Cluster Explorer

Results of the 2019 AWS Container Security Survey

Amazon EKS Price Reduction

Securing EKS Ingress With Contour And Let's Encrypt The GitOps Way

TMA Special: Connecting Taza Chocolate's Legacy Equipment to the Cloud

Using ALB Ingress Controller with Amazon EKS on Fargate

re:Cap part three – open source at re:Invent 2019