How To - Jenkins X_ a CI_CD solution for modern cloud applications on Kubernetes

How Jenkins-x can help in our CI/CD:

- Automate the installation, configuration and upgrade of Jenkins + other apps (helm (packaging), skaffold (building), nexus, monocular etc.) on K8s.
  Automates CI/CD for your applications on Kubernetes that means Docker images, Helm charts and Pipelines.
- Use GitOps to manage promotion between environments. i.e. Test -> Staging -> Production
- Lots of feedback
  e.g. commenting on the issues as they hit Staging + Production

What do we need for Jenkins-X:

Ingress controller (though that could probably just be vanilla helm stuff via the helm provider so probably not a custom Resource)

GitServers - the list of git services to use from inside Jenkins X. We need this to setup security/tokens and configure Jenkins etc. Attributes:

- kind (github/gitea/gitlab/bitbucket-cloud/bitbucket-server etc)
- name
- URL
- user name / API token to access it

Team a top level resource that contains child resources as follows. Attributes:

- name (the k8s service-style name and unique ID)
- label - the textual name
- git service name
- git organization name

User the users which get mapped to users inside k8s for RBAC. Attributes:

- name
- email address

Roles like a k8s Role but using an environment/namespace filter to define the namespaces it applies to. E.g. for all preview environments in team Cheese. Or all environments for Team cheese.

Team child resources
These resources live within a team:

Environments are the Dev / Staging / Production environments where apps run. Dev is the special one - it gets the jenkins-x-platform chart with team settings inside for things like build pack git URL,ref and branch patterns for CI/CD pipelines, locations of quickstarts and stuff. Attributes:

- name (the k8s service-style name like unique ID like staging or production
- label (the textual description like Production
- cluster API server URL for the multi-cluster case
- namespace name in the cluster
- git repository URL
- promotion kind (Auto/Manual)
- order (number) for the order the environments are processed during promotions

What is the difference between Jenkins and Jenkins X?

Unlike Jenkins, Jenkins X is opinionated and built to work better with technologies like Docker or Kubernetes. Having said that, Jenkins and Jenkins X are deeply related as everything that is done with Jenkins X can be done with Jenkins, using several plugins and integrations. However, Jenkins X simplifies everything, letting you harness the power of Jenkins 2.0 and using open source tools like Helm, Draft, Monocular, ChartMuseum, Nexus and Docker Registry to easily build cloud native applications.
In fact, it's this selection of tools and processes that make Jenkins X special and different from Jenkins and any other CI/CD solution. For instance, Jenkins X defines the process, while Jenkins adapts to whichever process are wanted or needed. Jenkins X adopts a CLI/API first approach, relies on configuration as code and embraces external tools (e.g., Helm, Monocular, etc). On the other hand, Jenkins has a UI first approach with configuration via UI, and everything heavily driven by internal plugins. Additionally, the Jenkins X Preview environments enable developers to collaboratively validate changes integrated into the codebase by creating a running system per Pull Request.

How do we setup Jenkins-x with Kubernetes:

Launch a t2.xlarge ec2 instance and then issue sudo yum update command to update the all the required apps.
Initially we need to create a cluster in Kubernetes (currently using eksctl)- so before that below components should be installed -

Installing Jenkins-x cli

To install Jenkins X on Linux, download the .tar file, and unarchive it in a directory where you can run the jx command.

Refer link ahead for more details on it https://jenkins-x.io/docs/getting-started/setup/install/

Download the jx binary archive using curl and pipe (|) the compressed archive to the tar command:

```
curl -L "https://github.com/jenkins-x/jx/releases/download/$(curl --silent https://api.github.com/repos/jenkins-x/jx/releases/latest | jq -r
'.tag_name')/jx-linux-amd64.tar.gz" | tar xzv "jx"
```

Or, if you don't have jq installed:

```
curl -L "https://github.com/jenkins-x/jx/releases/download/$(curl --silent "https://github.com/jenkins-x/jx/releases/latest" | sed 's#.tag/(.)\".*#\1#')/jx-li
nux-amd64.tar.gz" | tar xzv "jx"
```

Install the jx binary by moving it to a location which should be on your environments PATH, using the mv command:

```
sudo mv jx /usr/local/bin
```

Run jx version to make sure you're on the latest stable version

```
[root@ip-172-31-89-241 ~]# jx --version
2.0.976
```

To install or upgrade eksctl on Linux using curl

Refer link ahead to install it https://docs.aws.amazon.com/eks/latest/userguide/getting-started-eksctl.html

Download and extract the latest release of eksctl with the following command.

```
curl --silent --location "https://github.com/weaveworks/eksctl/releases/download/latest_release/eksctl_$(uname -s)_amd64.tar.gz" | tar xz -C /tmp
```

Move the extracted binary to /usr/local/bin.

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

Test that your installation was successful with the following command.

```
eksctl --version
```

Creating environment on UBUNTU machine:

Issue sudo apt-get update on the new Ubuntu instance to make the box up to date.

```
curl -L "https://github.com/jenkins-x/jx/releases/download/$(curl --silent "https://github.com/jenkins-x/jx/releases/latest" | sed 's#.tag/(.)\".*#\1#')/jx-li
nux-amd64.tar.gz" | tar xzv "jx"
```

```
sudo mv jx /usr/local/bin
```

```
jx --version
```

Installing aws-aunthenticator manually

https://docs.aws.amazon.com/eks/latest/userguide/install-aws-iam-authenticator.html

Installing helm manually

Installing helm manually in case if it is not automatically installed during cluster creation

Download helm compatible version as

```
wget https://get.helm.sh/helm-v2.14.0-linux-amd64.tar.gz
OR
wget https://storage.googleapis.com/kubernetes-helm/helm-v2.14.2-linux-amd64.tar.gz
```

Untar it and then do the following

```
[root@vbob ~]# tar -xzf helm-v2.14.0-linux-amd64.tar.gz
[root@vbob ~]# cp -rf linux-amd64/helm /usr/local/bin/
[root@vbob ~]# helm --version
Client: &version.Version{SemVer:"v2.14.0", GitCommit:"05811b84a3f93603dd6c2fcfe57944dfa7ab7fd0", GitTreeState:"clean"}
```

Configure aws configure and provide your access key and secret there to access eksctl service.

```
root@ip-172-31-84-101:~# aws configure
AWS Access Key ID [****************ZXHG]:
AWS Secret Access Key [****************qw9l]:
Default region name [us-east-1]:
Default output format [None]:
root@ip-172-31-84-101:~#
```

Installing Kubectl

Please refer the below link
https://kubernetes.io/docs/tasks/tools/install-kubectl/

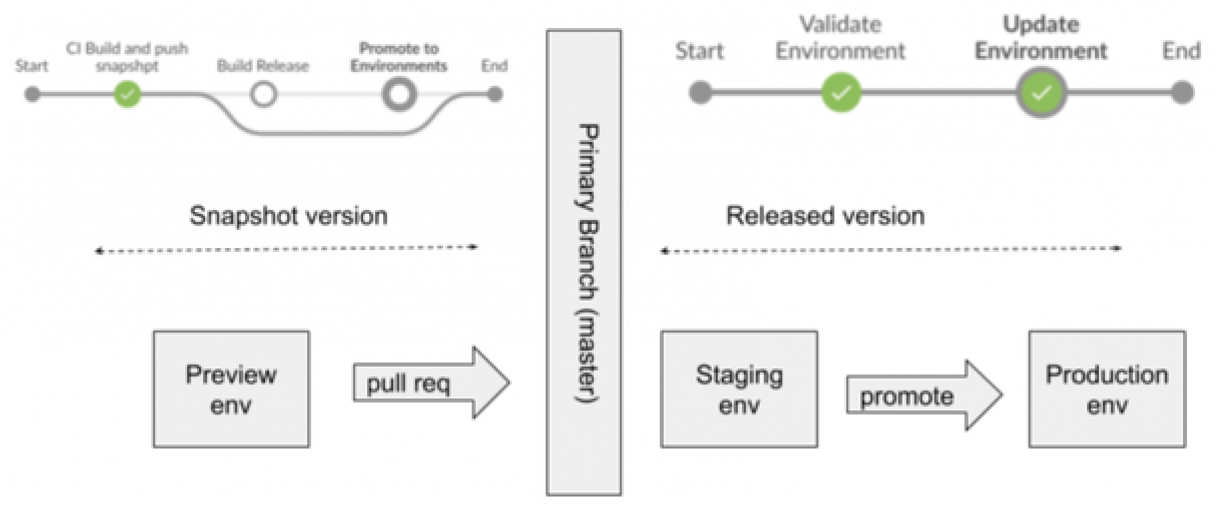
Installing gloo and knative

https://knative.dev/docs/install/knative-with-gloo/

```
[root@ip-172-31-93-239 ~]# jx create addon gloo
installing Knative CRDs...
installing Knative...
Knative successfully installed!
Starting Gloo installation...
Installing CRDs...
Preparing namespace and other pre-install tasks...
Installing...
Gloo was successfully installed!
```

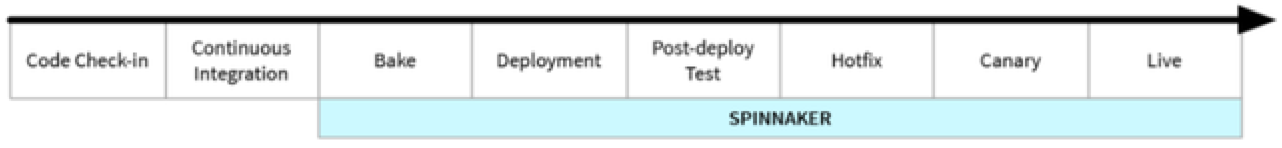
Jenkins-x architecture and E2E CICD flow:

Though Jenkins-X can be used on several cloud environments - here we are using AWS.
Amazon EKS and Jenkins-X installed on the cluster provide a continuous delivery platform that allows developers to focus on their applications.
Jenkins-X follows the best practices outlined by the Accelerate book and the State of DevOps report. This includes practices such as using Continuous Integration and Continuous Delivery, using trunk-based development, and using the cloud well.
Jenkins-X's **jx** command creates a git repository in Github for us, registers a Webhook to capture git push events and creates a Helm chart for our project. The Helm is used to package and perform installations and upgrade for our apps.
Jenkins-X manages deployment via source control. Each pull request is built and deployed to its own with an isolated preview environment (each promotion is actually a pull request to a per-environment repository-GitOps). Merges to master are automatically promoted to staging.
It uses Skaffold for image generation and CustomResourceDefinition(CRD) of Kubernetes to manage the pipeline activity.



Jenkins-X keeps environment configuration in source control. Deployments to master are done via pull request to the production configuration repository.
Compared with mature CI/CD tool Spinnaker, Jenkins-X is released recently (2018 March). So, we may want to use it for a pilot project in a smaller team as suggested in Spinnaker over Jenkins-X for Enterprise.
That's because Jenkins-X is quick to setup both CI and CD and it's quite straight forward (requires less intervention to pipeline configurations by providing strong defaults) to deploy kubernetes apps while the Spinnaker is mostly focused on the CD side providing various deployment

strategies (red/black, rolling, and canary), roll back on problems, approval, proper tearing down (thus, it gives us more freedom for CI tools, instead) as shown in the picture below.



Continuous Delivery with Spinnaker

Install binary jx command line tool

We will need to get the **jx** command line tool locally on our machine.
On a Mac we can use brew:

```
$ brew tap jenkins-x/jx $ brew install jx
==> Installing jx from jenkins-x/jx
==> Downloading https://github.com/jenkins-x/jx/releases/download/v1.3.560/jx-darwin-amd64.tar.gz
==> Downloading from https://github-production-release-asset-2e65be.s3.amazonaws.com/116400734/1ba97f00-e82f-11e8-8059-69feb9ff2b63?X-Amz-Al
############################################################################ 100.0%
/usr/local/Cellar/jx/1.3.560: 3 files, 82.6MB, built in 21 seconds
```

Linux:

```
$ mkdir -p ~/.jx/bin
$ curl -L https://github.com/jenkins-x/jx/releases/download/v1.3.560/jx-linux-amd64.tar.gz | tar xzv -C ~/.jx/bin
$ export PATH=$PATH:~/.jx/bin
$ echo 'export PATH=$PATH:~/.jx/bin' >> ~/.bashrc
```

To find out the available commands type:

```
$ jx
```

Or to get help on a specific command, say, create then type:

```
$ jx help create
```

After successful installation of **jx** client we should now be able to display the **jx** client version by executing the following command:

```
$ jx version
Using helmBinary helm with feature flag: none
NAME VERSION
jx 1.3.560
jenkins x platform 0.0.2871
Kubernetes cluster v1.10.3-eks
kubectl v1.11.3
helm client v2.11.0+g2e55dbe
helm server v2.11.0+g2e55dbe
git git version 2.15.1 (Apple Git-101)
```

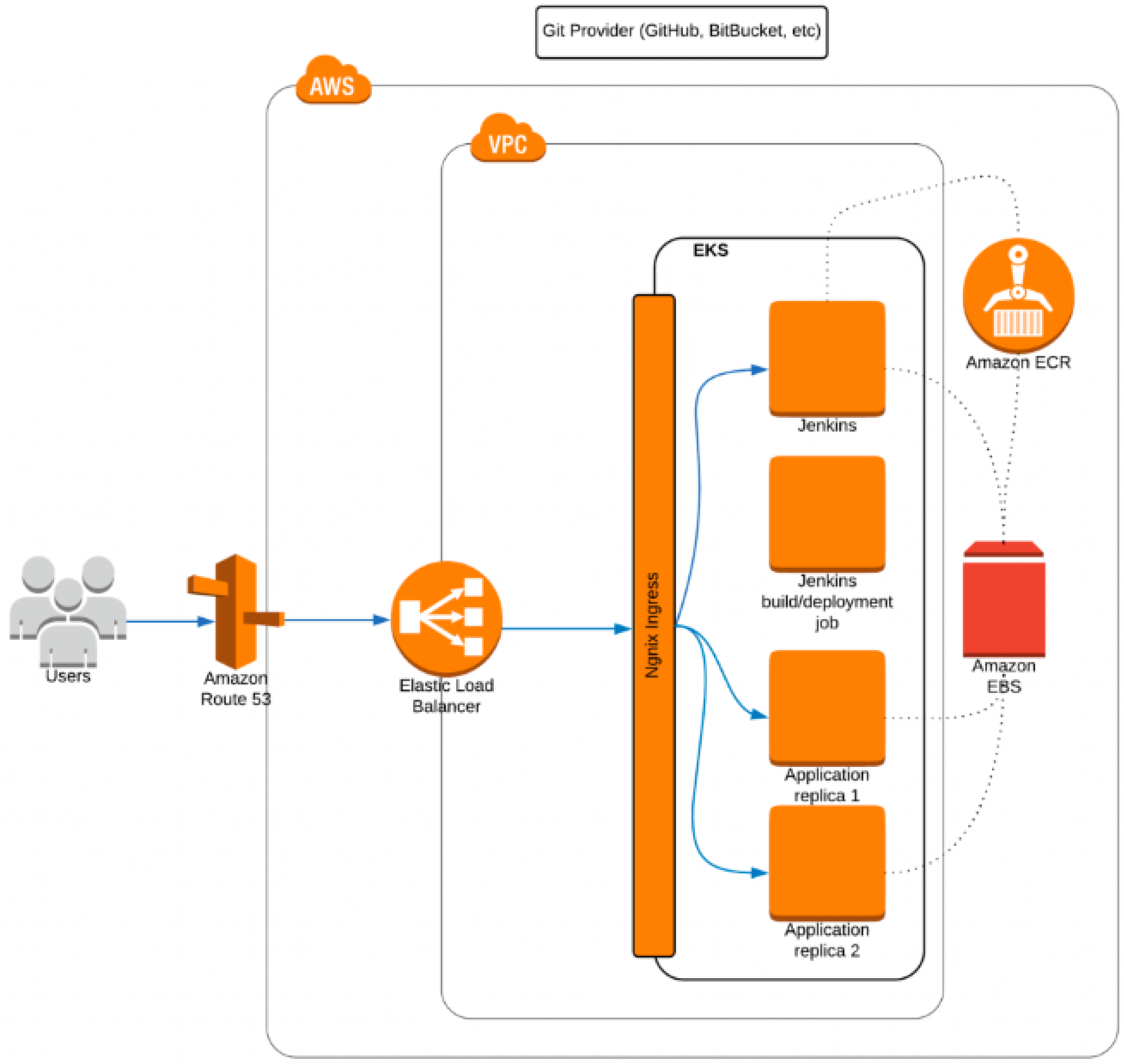
Creating a new Kubernetes cluster

We can create Kubernetes on AWS, Ggoogle, or Azure:

```
$ jx create cluster eks
$ jx create cluster gke
$ jx create cluster aks
```

Here, we will use AWS.
Now that we have **jx** client installed, we're ready to create the EKS cluster which will be used to run the Jenkins-X server, CI builds, and the application itself. Jenkins-X makes this task trivial by leveraging the power of the eksctl project.
We don't have to worry if we don't have eksctl, kubectl (Kubernetes client), Helm (package manager for Kubernetes), or the Heptio Authenticator for AWS installed: **jx** will detect anything that's missing and can automatically download and install it for us.
We can use the **jx** command line to create a new kubernetes cluster with **Jenkins-X** installed automatically.
While Jenkins recommends using Google Container Engine (GKE) for the best getting started experience, we're going to use AWS. Again, we have two options on AWS: either **kops** or **eks** which we'll use.

Before we start working with EKS, ensure that we have our AWS credentials set up. The Jenkins-X client is smart enough to retrieve AWS credentials from standard AWS cli locations, including environment variables or ~/.aws config files.



The diagram shows what we're going to build. Our goal is to have a EKS cluster deployed exposed to the outside world using Elastic Load Balancing (ELB) and Route 53 domain mapping (note that we'll use the nip.io service instead of a real domain).
Inside our cluster, we want to have a Jenkins server (reacting on the Git changes) which can start Kubernetes-based builds which end up as Docker images pushed into Elastic Container Registry (ECR).
Finally, we want Jenkins to deploy our application image into an EKS cluster and expose it via ingress to the outside world. All the persistence needs of our infrastructure and applications will be handled by Amazon services, for example using Amazon Elastic Block Store (Amazon EBS), which will be used by ECR or Kubernetes persistent volumes.
To take a quick way to get the taste of Jenkins-X, for Ingress, as mentioned earlier, we won't use DNS wildcard CNAME to point at our NLB hostname. Instead, we'll use an NLP and use one of the IP addresses of one of the availability zones as our domain via **$IP.ip**, which means we utilize only a single availability zone IP address by resolving the NLB host name to one of the availability zone IP addresses.
Note that this approach is not really intended for real production installations.

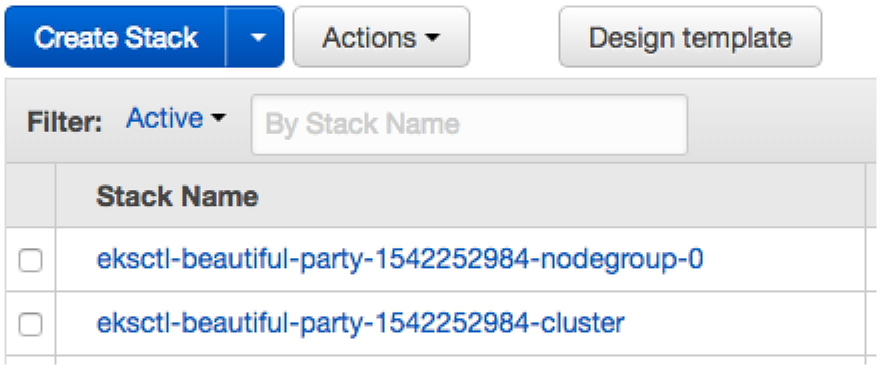
Now, we will download and use the **eksctl** tool to create a new EKS cluster, then it'll install Jenkins-X on top of the cluster (we can use "--skip-installation=true" option for provisioning the cluster only without installing Jenkins-X into it).
"jx create cluster eks command creates a new Kubernetes cluster on Amazon Web Services (AWS) using EKS, installing required local dependencies and provisions the Jenkins-X platform.

```
$ jx create cluster eks
```

```
? Missing required dependencies, deselect to avoid auto installing: [Use arrows to move, type to filter]
> [x] eksctl
[x] heptio-authenticator-aws
[x] helm
```

Hit "return":

```
Missing required dependencies, deselect to avoid auto installing: eksctl, heptio-authenticator-aws, helm
Downloading https://github.com/weaveworks/eksctl/releases/download/0.1.5/eksctl_darwin_amd64.tar.gz to
/Users/kihyuckhong/.jx/bin/eksctl.tar.gz...
Downloaded /Users/kihyuckhong/.jx/bin/eksctl.tar.gz
Downloading https://amazon-eks.s3-us-west-2.amazonaws.com/1.10.3/2018-06-05/bin/darwin/amd64/heptio-authenticator-aws to
/Users/kihyuckhong/.jx/bin/heptio-authenticator-aws...
Downloaded /Users/kihyuckhong/.jx/bin/heptio-authenticator-aws
Downloading https://storage.googleapis.com/kubernetes-helm/helm-v2.11.0-darwin-amd64.tar.gz to /Users/kihyuckhong/.jx/bin/helm.tgz...
Downloaded /Users/kihyuckhong/.jx/bin/helm.tgz
Using helmBinary helm with feature flag: none
Creating EKS cluster - this can take a while so please be patient...
You can watch progress in the CloudFormation console: https://console.aws.amazon.com/cloudformation/
...
```



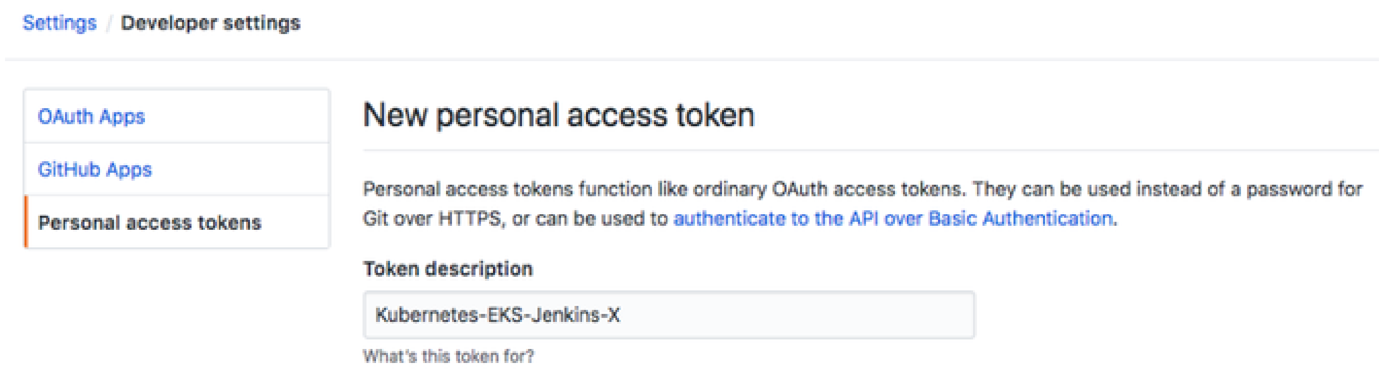
eksctl-beautiful-party-1542252984-nodegroup-0
eksctl-beautiful-party-1542252984-cluster

Creating cluster on EKS takes much longer than on Google GKE.
There are more outputs:

```
Initializing cluster...
Using helmBinary helm with feature flag: none
Namespace jx created
Storing the kubernetes provider eks in the TeamSettings
Updated the team settings in namespace jx
? Please enter the name you wish to use with git: Einsteinish
? Please enter the email address you wish to use with git: kihyuck.hong@gmail.com
Git configured for user: Einsteinish and email kihyuck.hong@gmail.com
Trying to create ClusterRoleBinding kihyuck-hong-gmail-com-cluster-admin-binding for role: cluster-admin for user kihyuck.hong@gmail.com
clusterrolebindings.rbac.authorization.k8s.io "kihyuck-hong-gmail-com-cluster-admin-binding" not found
Created ClusterRoleBinding kihyuck-hong-gmail-com-cluster-admin-binding
Using helm2
Configuring tiller
Created ServiceAccount tiller in namespace kube-system
Trying to create ClusterRoleBinding tiller for role: cluster-admin and ServiceAccount: kube-system/tiller
Created ClusterRoleBinding tiller
Initialising helm using ServiceAccount tiller in namespace kube-system
Using helmBinary helm with feature flag: none
Waiting for tiller-deploy to be ready in tiller namespace kube-system
helm installed and configured
? No existing ingress controller found in the kube-system namespace, shall we install one? Yes
Using helm values file: /var/folders/3s/42rgkz495lj4ydjgnqhxn_400000gn/T/ing-values-920523981
Installing using helm binary: helm
Waiting for external loadbalancer to be created and update the nginx-ingress-controller service in kube-system namespace
External loadbalancer created
Waiting to find the external host name of the ingress controller Service in namespace kube-system with name jxing-nginx-ingress-controller
On AWS we recommend using a custom DNS name to access services in your Kubernetes cluster to ensure you can use all of your Availability
Zones
If you do not have a custom DNS name you can use yet you can register a new one here: https://console.aws.amazon.com/route53/home?#D
omainRegistration:
? Would you like to register a wildcard DNS ALIAS to point at this ELB address? [? for help] (Y/n) n
? Would you like wait and resolve this address to an IP address and use it for the domain? Yes
Waiting for a82b8afa0e86411e8b74e166e925a7f8-97772dc3463ccff9.elb.us-east-1.amazonaws.com to be resolvable to an IP address...
retrying after error: Address cannot be resolved yet a82b8afa0e86411e8b74e166e925a7f8-97772dc3463ccff9.elb.us-east-1.amazonaws.com
.
```

```
a82b8afa0e86411e8b74e166e925a7f8-97772dc3463ccff9.elb.us-east-1.amazonaws.com resolved to IP 54.172.116.89
You can now configure a wildcard DNS pointing to the new loadbalancer address 54.172.116.89
If you do not have a custom domain setup yet, Ingress rules will be set for magic dns nip.io.
Once you have a customer domain ready, you can update with the command jx upgrade ingress --cluster
If you don't have a wildcard DNS setup then setup a new CNAME and point it at: 54.172.116.89.nip.io then use the DNS domain in the next input...
? Domain 54.172.116.89.nip.io
nginx ingress controller installed and configured
Lets set up a Git username and API token to be able to perform CI/CD
? GitHub user name: Einsteinish
To be able to create a repository on GitHub we need an API Token
Please click this URL https://github.com/settings/tokens/new?scopes=repo,read:user,read:org,user:email,write:repo_hook,delete_repo
Then COPY the token and enter in into the form below:
```

We can generate the GitHub API token with the scopes given as the parameters (scopes=repo,read:user,read:org,user:email,write:repo_hook,delete_repo):


Settings / Developer settings

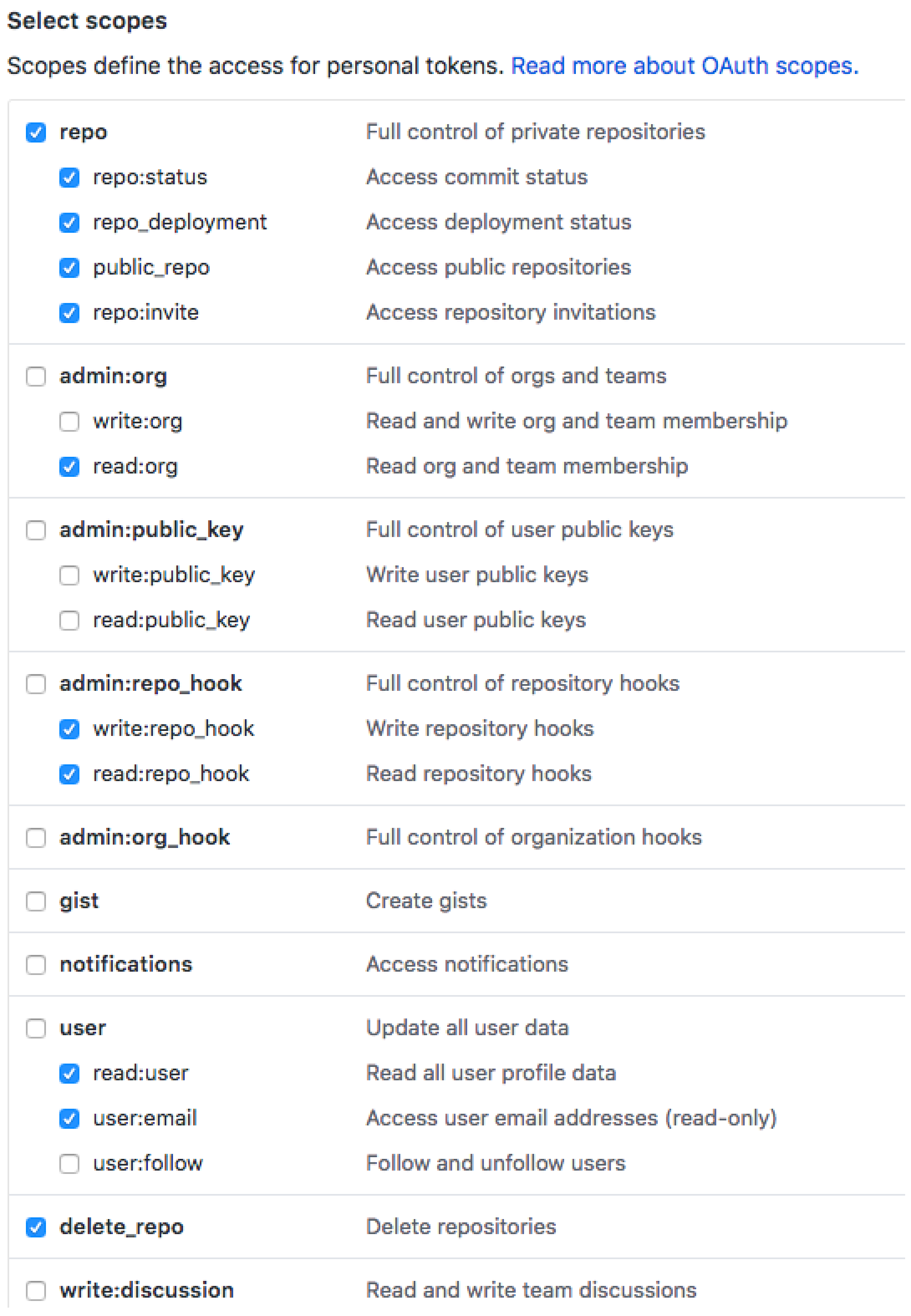
OAuth Apps
GitHub Apps
Personal access tokens

New personal access token

Personal access tokens function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to authenticate to the API over Basic Authentication.

Token description: Kubernetes-EKS-Jenkins-X

What's this token for?



Select scopes

Scopes define the access for personal tokens. Read more about OAuth scopes.

✓ **repo**	Full control of private repositories
✓ repo:status	Access commit status
✓ repo_deployment	Access deployment status
✓ public_repo	Access public repositories
✓ repo:invite	Access repository invitations
☐ **admin:org**	Full control of orgs and teams
☐ write:org	Read and write org and team membership
✓ read:org	Read org and team membership
☐ **admin:public_key**	Full control of user public keys
☐ write:public_key	Write user public keys
☐ read:public_key	Read user public keys
☐ **admin:repo_hook**	Full control of repository hooks
✓ write:repo_hook	Write repository hooks
✓ read:repo_hook	Read repository hooks
☐ **admin:org_hook**	Full control of organization hooks
☐ **gist**	Create gists
☐ **notifications**	Access notifications
☐ **user**	Update all user data
✓ read:user	Read all user profile data
✓ user:email	Access user email addresses (read-only)
☐ user:follow	Follow and unfollow users
✓ **delete_repo**	Delete repositories
☐ **write:discussion**	Read and write team discussions

A personal access token (Kubernetes-EKS-Jenkins-X) with delete_repo, read:org, read:user, repo, user:email, and write:repo_hook scopes was recently added to our account. We may want to visit https://github.com/settings/tokens for more information.
To see this security events for our GitHub account, visit https://github.com/settings/security
Type in the newly created GitHub token.

```
? API Token: ****************************************
Updated the team settings in namespace jx
```

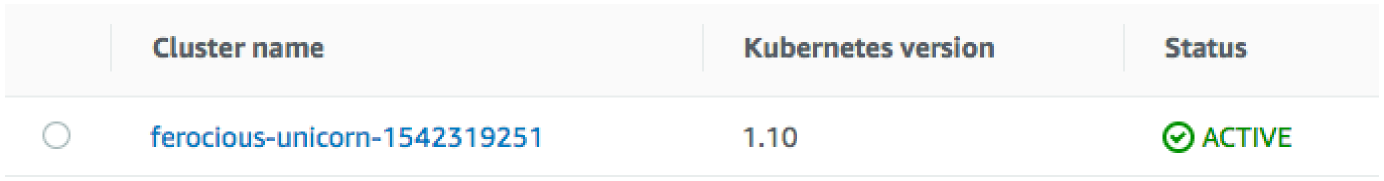
```
Cloning the Jenkins X cloud environments repo to /Users/kihyuckhong/.jx/cloud-environments
Enumerating objects: 1210, done.
Total 1210 (delta 0), reused 0 (delta 0), pack-reused 1210
No default password set, generating a random one
Creating secret jx-install-config in namespace jx
Generated helm values /Users/kihyuckhong/.jx/extraValues.yaml
? Select Jenkins installation type: [Use arrows to move, type to filter]
Serverless Jenkins
> Static Master Jenkins
```

Here is the final output:

```
? Select Jenkins installation type: Static Master Jenkins
Installing Jenkins X platform helm chart from: /Users/kihyuckhong/.jx/cloud-environments/env-eks
Installing jx into namespace jx
Waiting for tiller pod to be ready, service account name is tiller, namespace is jx, tiller namespace is kube-system
Waiting for cluster role binding to be defined, named tiller-role-binding in namespace jx
tiller cluster role defined: cluster-admin in namespace jx
tiller pod running
Adding values file /Users/kihyuckhong/.jx/cloud-environments/env-eks/myvalues.yaml
Adding values file /Users/kihyuckhong/.jx/gitSecrets.yaml
Adding values file /Users/kihyuckhong/.jx/adminSecrets.yaml
Adding values file /Users/kihyuckhong/.jx/extraValues.yaml
Adding values file /Users/kihyuckhong/.jx/cloud-environments/env-eks/secrets.yaml
waiting for install to be ready, if this is the first time then it will take a while to download images
Jenkins X deployments ready in namespace jx

********************************************************
NOTE: Your admin password is: swisherpepper
********************************************************

Getting Jenkins API Token
Using url http://jenkins.jx.54.147.194.9.nip.io/me/configure
Generating the API token...
Created user admin API Token for Jenkins server jenkins.jx.54.147.194.9.nip.io at http://jenkins.jx.54.147.194.9.nip.io
Updating Jenkins with new external URL details http://jenkins.jx.54.147.194.9.nip.io
Creating default staging and production environments
Using Git provider GitHub at https://github.com

About to create repository environment-sagepinto-staging on server https://github.com with user Einsteinish

Creating repository Einsteinish/environment-sagepinto-staging
Creating Git repository Einsteinish/environment-sagepinto-staging
Pushed Git repository to https://github.com/Einsteinish/environment-sagepinto-staging
Creating staging Environment in namespace jx
Created environment staging
Namespace jx-staging created
Updated the team settings in namespace jx
Created Jenkins Project: http://jenkins.jx.54.147.194.9.nip.io/job/Einsteinish/job/environment-sagepinto-staging/
Note that your first pipeline may take a few minutes to start while the necessary images get downloaded!
Creating GitHub webhook for Einsteinish/environment-sagepinto-staging for url http://jenkins.jx.54.147.194.9.nip.io/github-webhook/
Using Git provider GitHub at https://github.com

About to create repository environment-sagepinto-production on server https://github.com with user Einsteinish

Creating repository Einsteinish/environment-sagepinto-production
Creating Git repository Einsteinish/environment-sagepinto-production
Pushed Git repository to https://github.com/Einsteinish/environment-sagepinto-production
Creating production Environment in namespace jx
Created environment production
Namespace jx-production created
Updated the team settings in namespace jx
Created Jenkins Project: http://jenkins.jx.54.147.194.9.nip.io/job/Einsteinish/job/environment-sagepinto-production/
Note that your first pipeline may take a few minutes to start while the necessary images get downloaded!
Creating GitHub webhook for Einsteinish/environment-sagepinto-production for url http://jenkins.jx.54.147.194.9.nip.io/github-webhook/
Jenkins X installation completed successfully

********************************************************
NOTE: Your admin password is: swisherpepper
********************************************************


Your Kubernetes context is now set to the namespace: jx
```

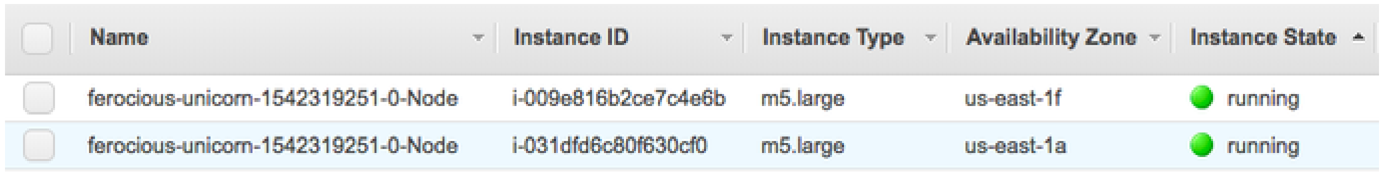
```
To switch back to your original namespace use: jx ns default
For help on switching contexts see: https://jenkins-x.io/developing/kube-context/
To import existing projects into Jenkins: jx import
To create a new Spring Boot microservice: jx create spring -d web -d actuator
To create a new microservice from a quickstart: jx create quickstart
$
```

Now we will be able to see our cluster:



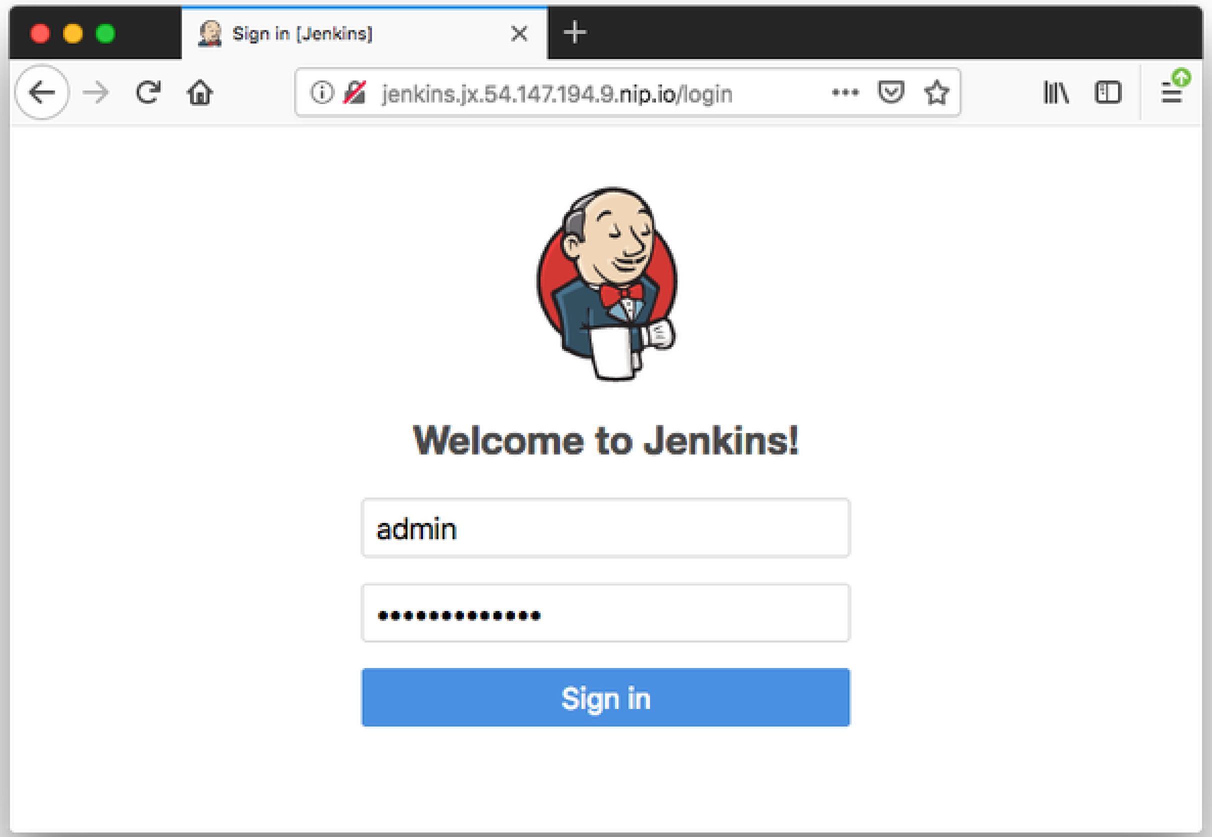
	Cluster name	Kubernetes version	Status
○	ferocious-unicorn-1542319251	1.10	ACTIVE

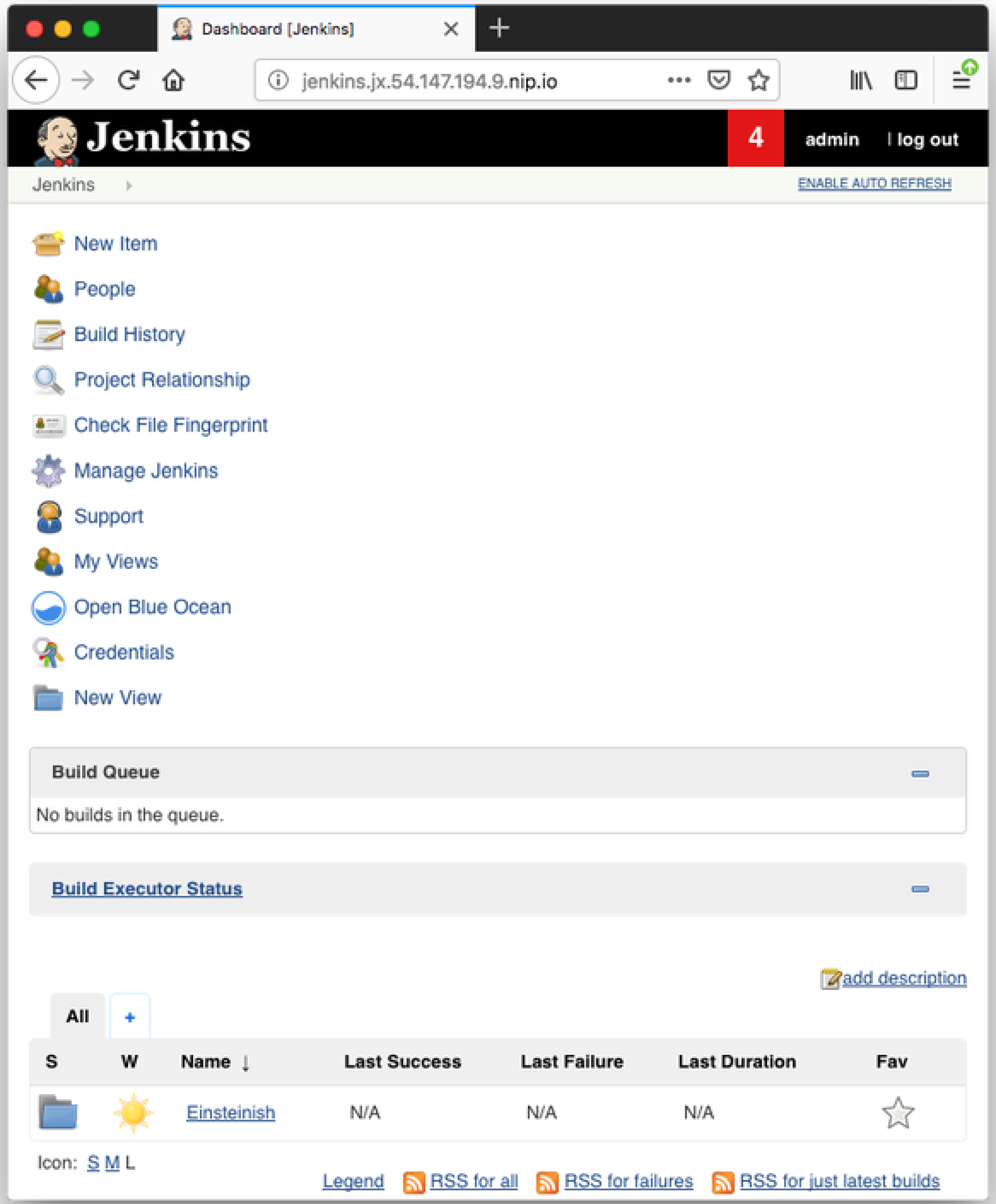
We should also be able to see the EC2 worker nodes associated with our cluster:



	Name	Instance ID	Instance Type	Availability Zone	Instance State
	ferocious-unicorn-1542319251-0-Node	i-009e816b2ce7c4e6b	m5.large	us-east-1f	● running
	ferocious-unicorn-1542319251-0-Node	i-031dfd6c80f630cf0	m5.large	us-east-1a	● running

Jenkins-X with nip.io address:


Sign in [Jenkins]
jenkins.jx.54.147.194.9.nip.io/login
Welcome to Jenkins!
admin
••••••••••••••
Sign in



Build Queue

No builds in the queue.

Build Executor Status

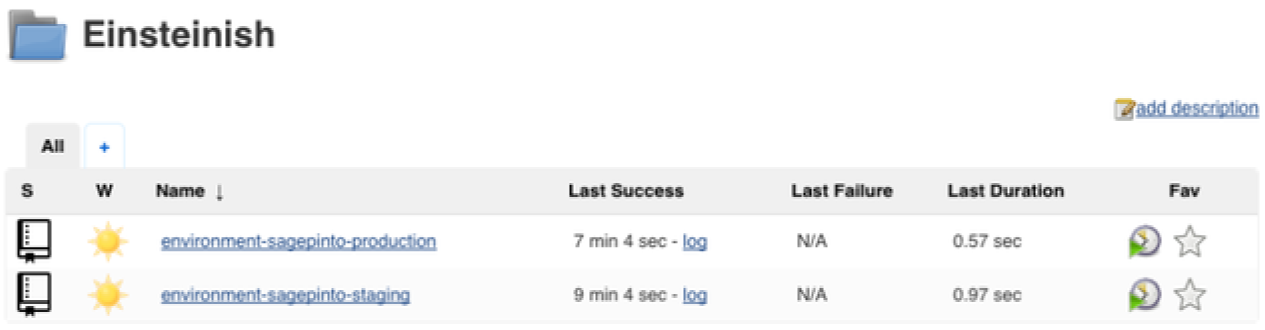
add description

All +

S	W	Name ↓	Last Success	Last Failure	Last Duration	Fav
		Einsteinish	N/A	N/A	N/A	☆

Icon: S M L

Legend RSS for all RSS for failures RSS for just latest builds



Einsteinish

add description

All +

S	W	Name ↓	Last Success	Last Failure	Last Duration	Fav
	☀	environment-sagepinto-production	7 min 4 sec - log	N/A	0.57 sec	☆
	☀	environment-sagepinto-staging	9 min 4 sec - log	N/A	0.97 sec	☆

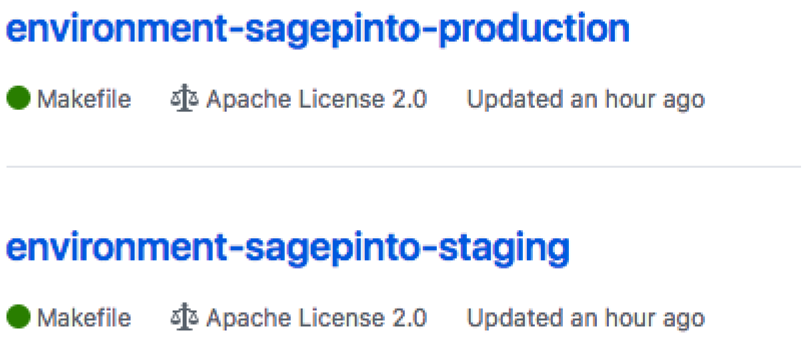
$ jx get env

```
NAME LABEL KIND PROMOTE NAMESPACE ORDER CLUSTER SOURCE REF PR
dev Development Development Never jx 0
staging Staging Permanent Auto jx-staging 100 https://github.com/Einsteinish/environment-sagepinto-staging.git
production Production Permanent Manual jx-production 200 https://github.com/Einsteinish/environment-sagepinto-production.git
```

$ jx get app

```
No applications found in environments staging, production.
```

We can also navigate to GitHub and see that Jenkins X provisioned the projects:



environment-sagepinto-production

● Makefile Apache License 2.0 Updated an hour ago

environment-sagepinto-staging

● Makefile Apache License 2.0 Updated an hour ago

Below is sample **Helm chart definitions** as just built by Jenkins (**environment-sagepinto-production/env/values.yaml**):

```
expose:
  Args:
    - --v
    - 4
  Annotations:
    helm.sh/hook: post-install,post-upgrade
    helm.sh/hook-delete-policy: hook-succeeded
cleanup:
  Args:
    - --cleanup
  Annotations:
    helm.sh/hook: pre-delete
    helm.sh/hook-delete-policy: hook-succeeded
expose:
  config:
    domain: 54.147.194.9.nip.io
    exposer: Ingress
    http: "true"
    tlsacme: "false"
    pathMode: ""
  Annotations:
    helm.sh/hook: post-install,post-upgrade
    helm.sh/hook-delete-policy: hook-succeeded
jenkins:
  Servers:
    Global:
      EnvVars:
        DOCKER_REGISTRY: 526262051452.dkr.ecr.us-east-1.amazonaws.com
        TILLER_NAMESPACE: kube-system
```

Spring Boot Micro service

Let's create a bare-bones Spring Boot app from Cloud Shell.
During the process of creating the Spring Boot app, jx will ask us about the project name, the language you want to use for the project, Maven coordinates, etc.

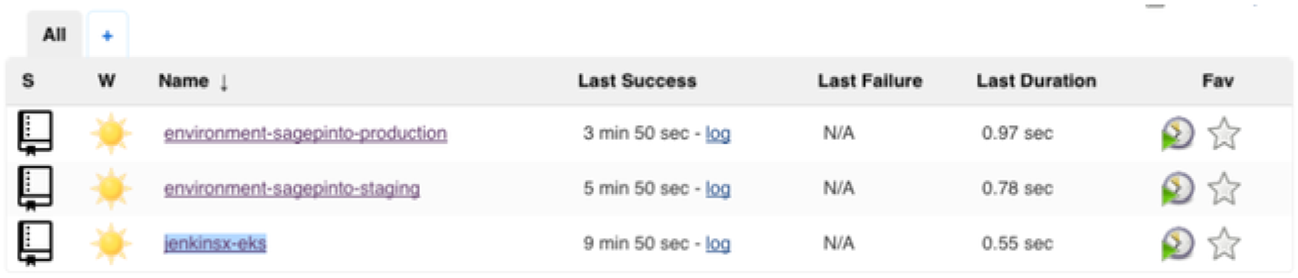
```
$ jx create spring -d web -d actuator
? Language: java
? Group: com.example
? Artifact: jenkinsx-eks
Created Spring Boot project at /Users/kihyuckhong/Documents/TEST/SpringJenkinsX/demo
```

```
No username defined for the current Git server!
? Do you wish to use Einsteinish as the Git user name: Yes
The directory /Users/kihyuckhong/Documents/TEST/SpringJenkinsX/demo is not yet using git
? Would you like to initialise git now? Yes
? Commit message: Initial import
Git repository created
selected pack: /Users/kihyuckhong/.jx/draft/packs/github.com/jenkins-x/draft-packs/packs/maven
existing Dockerfile, Jenkinsfile and charts folder found so skipping 'draft create' step
Using Git provider GitHub at https://github.com

About to create repository demo on server https://github.com with user Einsteinish
? Which organisation do you want to use? Einsteinish
? Enter the new repository name: jenkinsx-eks

Creating repository Einsteinish/jenkinsx-eks
Pushed Git repository to https://github.com/Einsteinish/jenkinsx-eks
Let's ensure that we have an ECR repository for the Docker image einsteinish/jenkins-x-eks
Created ECR repository: 526262051452.dkr.ecr.us-east-1.amazonaws.com/einsteinish/jenkins-x-eks
Updated the team settings in namespace jx
Created Jenkins Project: http://jenkins.jx.54.147.194.9.nip.io/job/Einsteinish/job/jenkinsx-eks/
Watch pipeline activity via: jx get activity -f jenkinsx-eks -w
Browse the pipeline log via: jx get build logs Einsteinish/jenkinsx-eks/master
Open the Jenkins console via jx console
You can list the pipelines via: jx get pipelines
When the pipeline is complete: jx get applications
For more help on available commands see: https://jenkins-x.io/developing/browsing/
Note that your first pipeline may take a few minutes to start while the necessary images get downloaded!
Creating GitHub webhook for Einsteinish/jenkinsx-eks for url http://jenkins.jx.54.147.194.9.nip.io/github-webhook/
```

Now we can see that our application has been created on our local laptop, pushed into the remote GitHub repository, and added into the Jenkins X CD pipeline.
Let's check that our app is in Jenkins X CD pipeline with **jx get pipe** command:

```
$ jx get pipe

Name URL LAST_BUILD STATUS DURATION
Einsteinish/environment-sagepinto-production/master http://jenkins.jx.54.147.194.9.nip.io/job/Einsteinish/job/environment-sagepinto-production/job/master/ #1 SUCCESS 177.955µs
Einsteinish/environment-sagepinto-staging/PR-1 http://jenkins.jx.54.147.194.9.nip.io/job/Einsteinish/job/environment-sagepinto-staging/job/PR-1/ #1 SUCCESS 83.107µs
Einsteinish/environment-sagepinto-staging/master http://jenkins.jx.54.147.194.9.nip.io/job/Einsteinish/job/environment-sagepinto-staging/job/master/ #2 SUCCESS 71.465µs
Einsteinish/jenkinsx-eks/master http://jenkins.jx.54.147.194.9.nip.io/job/Einsteinish/job/jenkinsx-eks/job/master/ #1 SUCCESS 382.321µs
```

The command shows the list of pipelines registered in Jenkins X. As we can see, we have dedicated pipelines for changes that should be applied to staging and production environments.
We can also use the Jenkins UI to see the pipeline progress:

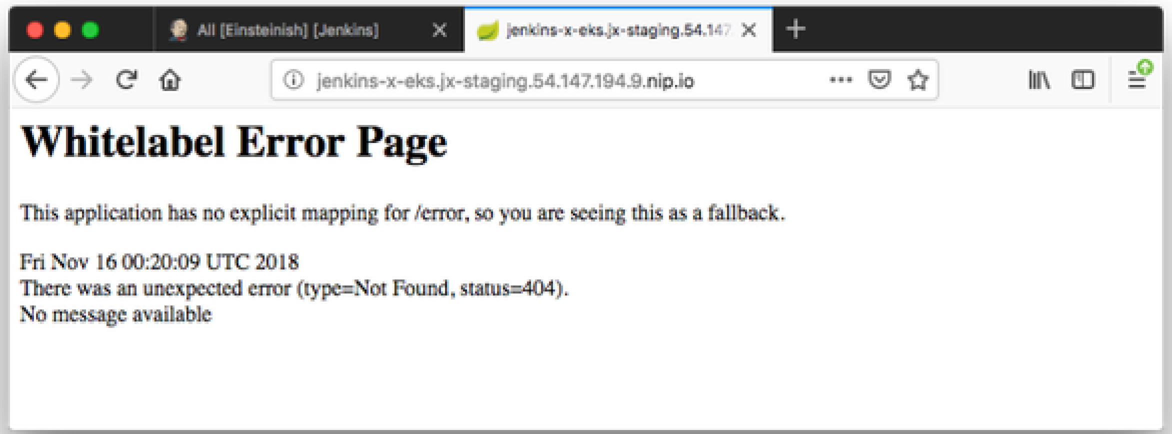


S	W	Name ↓	Last Success	Last Failure	Last Duration	Fav
		environment-sagepinto-production	3 min 50 sec - log	N/A	0.97 sec	
		environment-sagepinto-staging	5 min 50 sec - log	N/A	0.78 sec	
		jenkinsx-eks	9 min 50 sec - log	N/A	0.55 sec	

We can see this is listed as an app:

```
$ jx get app
APPLICATION STAGING PODS URL PRODUCTION PODS URL
jenkins-x-eks 0.0.1 1/1 http://jenkins-x-eks.jx-staging.54.147.194.9.nip.io
```

Jenkins X has now built and deployed our application into the staging environment. Let's use the URL of the application provided by the jx command output (http://jenkins-x-eks.jx-staging.54.147.194.9.nip.io) to see if it is up and running:


All [Einsteinish] [Jenkins]
jenkins-x-eks.jx-staging.54.147
jenkins-x-eks.jx-staging.54.147.194.9.nip.io

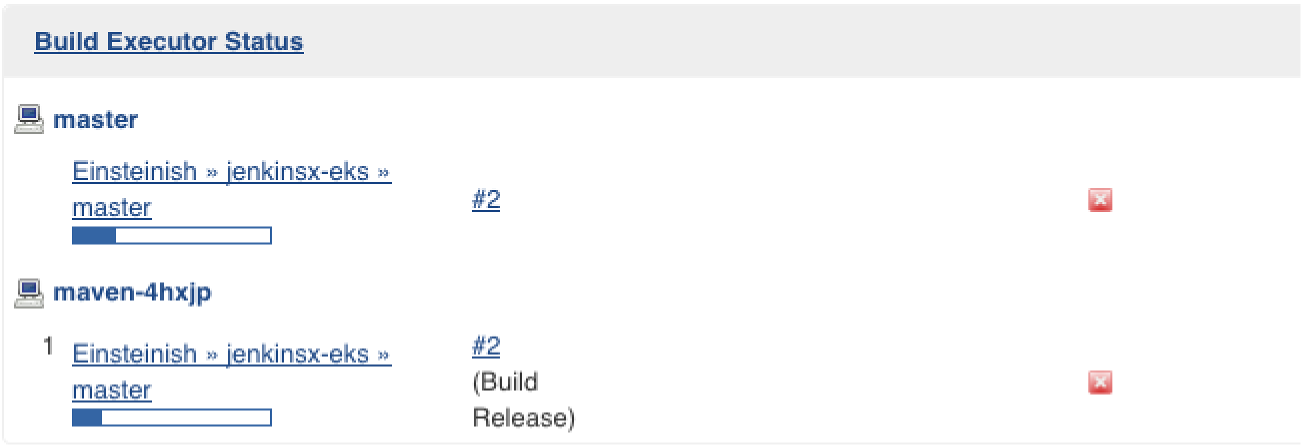
Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Fri Nov 16 00:20:09 UTC 2018
There was an unexpected error (type=Not Found, status=404).
No message available

If we can see the Spring Whitelabel error message, our application has been successfully deployed into our EKS cluster and exposed via the **Ingr ess** controller.

Now let's modify README file in our application.



Build Executor Status

master			
	Einsteinish » jenkinsx-eks » master	#2	✗
maven-4hxjp			
1	Einsteinish » jenkinsx-eks » master	#2 (Build Release)	✗

As we can see from the picture above, Jenkins X is rebuilding the app and and then redeploy the project. Now we see the app version has been changed to 0.0.2 from 0.0.1:

```
$ jx get app
APPLICATION STAGING PODS URL PRODUCTION PODS URL
jenkins-x-eks 0.0.2 1/1 http://jenkins-x-eks.jx-staging.54.147.194.9.nip.io
```

if you are unable to see the apps using the above command please try running as below
jx get applications

Or

jx open –e staging
jx open –e production