

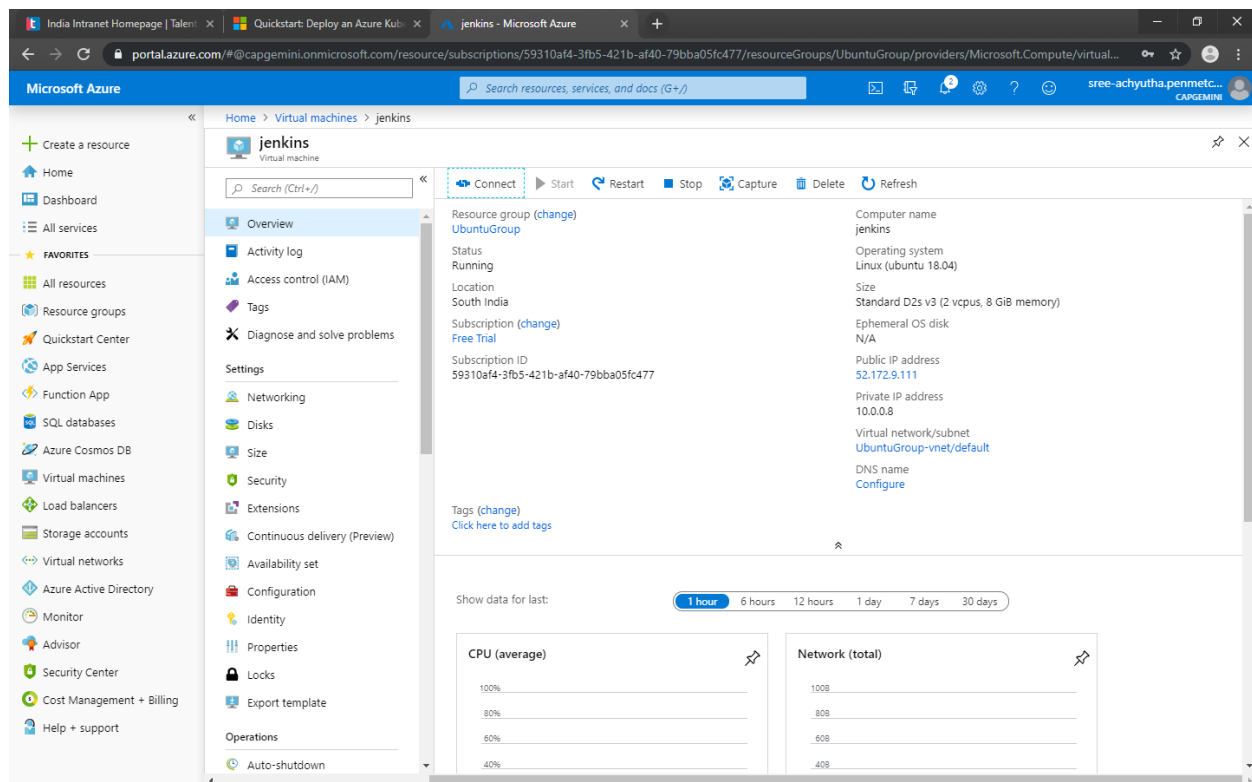
1. Jenkins on Azure

1.1 What is Jenkins?

Jenkins is a free and open source automation server written in Java. Jenkins helps to automate the non-human part of the software development process, with continuous integration and facilitating technical aspects of continuous delivery. Jenkins can be installed through native system packages, Docker, or even run standalone by any machine with a Java Runtime Environment (JRE) installed.

2. Jenkins Installation

Make sure you've Ubuntu VM ready on Azure



Azure Ubuntu VM Deployment



Connected to Ubuntu VM using Windows PowerShell

Before installing Jenkins, first you need to install few pre-required software in order to make Jenkins work flawlessly.

2.1 Java Installation

You have to install Java first before installing the Jenkins. Since Jenkins runs on Java, you need either latest version of Java Development Kit (JDK) or Java Runtime Environment (JRE).

You can use the following command to install JDK:

```
sudo apt install openjdk-8-jdk
```

After installing, you can verify by using this command:

```
java -version
```

```
capgemini@jenkins:~$ java -version
openjdk version "1.8.0_222"
OpenJDK Runtime Environment (build 1.8.0_222-8u222-b10-1ubuntu1~18.04.1-b10)
OpenJDK 64-Bit Server VM (build 25.222-b10, mixed mode)
```

2.2 Maven Installation

Apache Maven is a software project management and comprehension tool. Based on the concept of a project object model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information.

Installing Maven on Ubuntu VM is fairly easy. Just use the following command below:

```
sudo apt install maven
```

After installing, you can verify the maven version by using “mvn -version”

```
capgemini@jenkins:~$ mvn -version
Apache Maven 3.6.0
Maven home: /usr/share/maven
Java version: 1.8.0_222, vendor: Private Build, runtime: /usr/lib/jvm/java-8-openjdk-amd64/jre
Default locale: en, platform encoding: UTF-8
OS name: "linux", version: "5.0.0-1018-azure", arch: "amd64", family: "unix"
```

2.3 Jenkins Installation

The version of Jenkins included with the default Ubuntu packages is often behind the latest available version from the project itself. To take advantage of the latest fixes and features, you can use the project-maintained packages to install Jenkins.

1. First we need to add a repository key to the system:

```
wget -q -O - https://pkg.jenkins.io/debian/jenkins.io.key | sudo apt-key add
```

—

2. When you get the response as “OK”, then append the Debian package repository address to the server's sources.list:

```
sudo sh -c 'echo deb http://pkg.jenkins.io/debian-stable binary/ >
/etc/apt/sources.list.d/jenkins.list'
```

3. Now you need to update it so that apt will use the new repository

```
sudo apt update
```

4. Finally, you can install Jenkins now:

```
sudo apt install jenkins
```

```
capgemini@jenkins:~$ sudo apt install jenkins
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  daemon
The following NEW packages will be installed:
  daemon jenkins
0 upgraded, 2 newly installed, 0 to remove and 44 not upgraded.
Need to get 78.0 MB of archives.
After this operation, 78.6 MB of additional disk space will be used.
Do you want to continue? [Y/n]
```

Since Jenkins will run on port “8080”, we need to configure the firewall to allow 8080 connections. In order to do that, follow the commands below

1. By default, Jenkins runs on port 8080, so let's open that port using ufw:

```
sudo ufw allow 8080
```

2. Check ufw's status to confirm the new rules:

```
Sudo ufw status
```

```
capgemini@jenkins:~$ sudo ufw status
Status: active
```

To	Action	From
--	-----	----
8080	ALLOW	Anywhere
OpenSSH	ALLOW	Anywhere
8080 (v6)	ALLOW	Anywhere (v6)
OpenSSH (v6)	ALLOW	Anywhere (v6)

3. Just in case, if the output of above command is “inactive”, use the following commands below to open up firewall.
 - a. `sudo ufw allow OpenSSH`
 - b. `sudo ufw enable`

Now, check whether Jenkins is working on port 8080 by using the IP address of your machine.
using your server domain name or IP address: `http://your_server_ip_or_domain:8080`

You'll be getting the following output given below

Getting Started

Unlock Jenkins

To ensure Jenkins is securely set up by the administrator, a password has been written to the log ([not sure where to find it?](#)) and this file on the server:

```
/var/lib/jenkins/secrets/initialAdminPassword
```

Please copy the password from either location and paste it below.

Administrator password

Continue

2.4 Accessing Jenkins

Since you've installed Jenkins and successfully instantiated it, you need an Admin Password

Getting Started

Unlock Jenkins

To ensure Jenkins is securely set up by the administrator, a password has been written to the log (not sure where to find it?) and this file on the server:

```
/var/lib/jenkins/secrets/initialAdminPassword
```

Please copy the password from either location and paste it below.

Administrator password

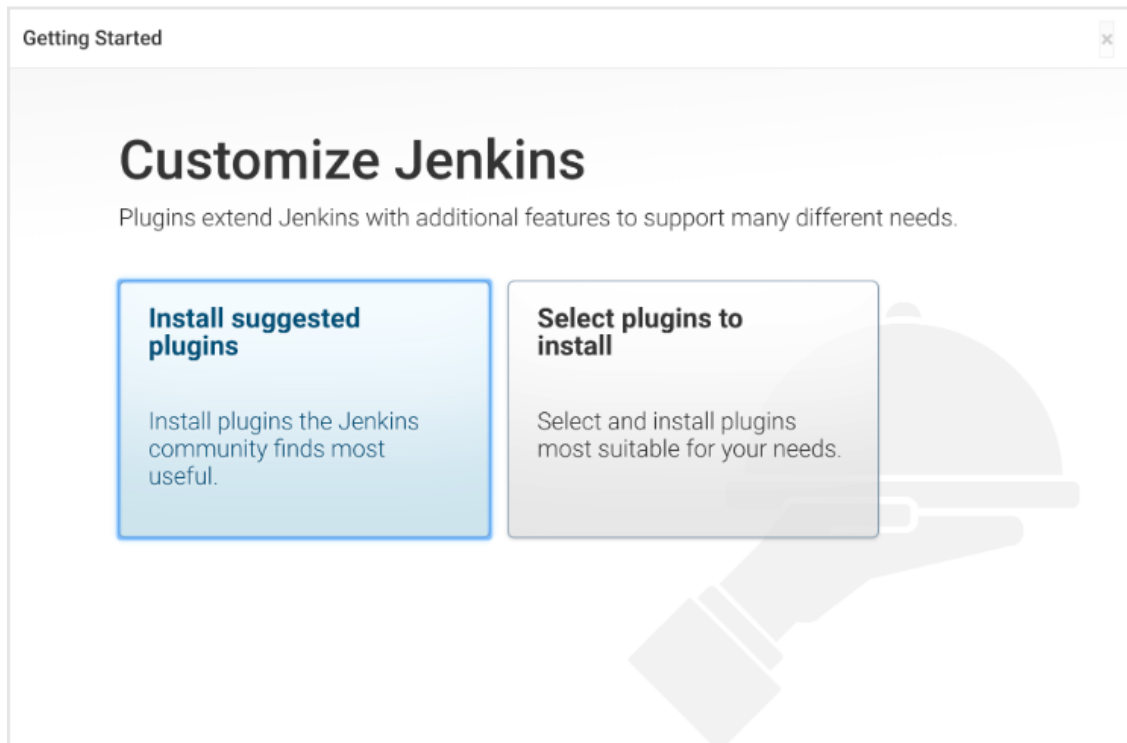
Continue

You can get the Admin Password by using the command below:

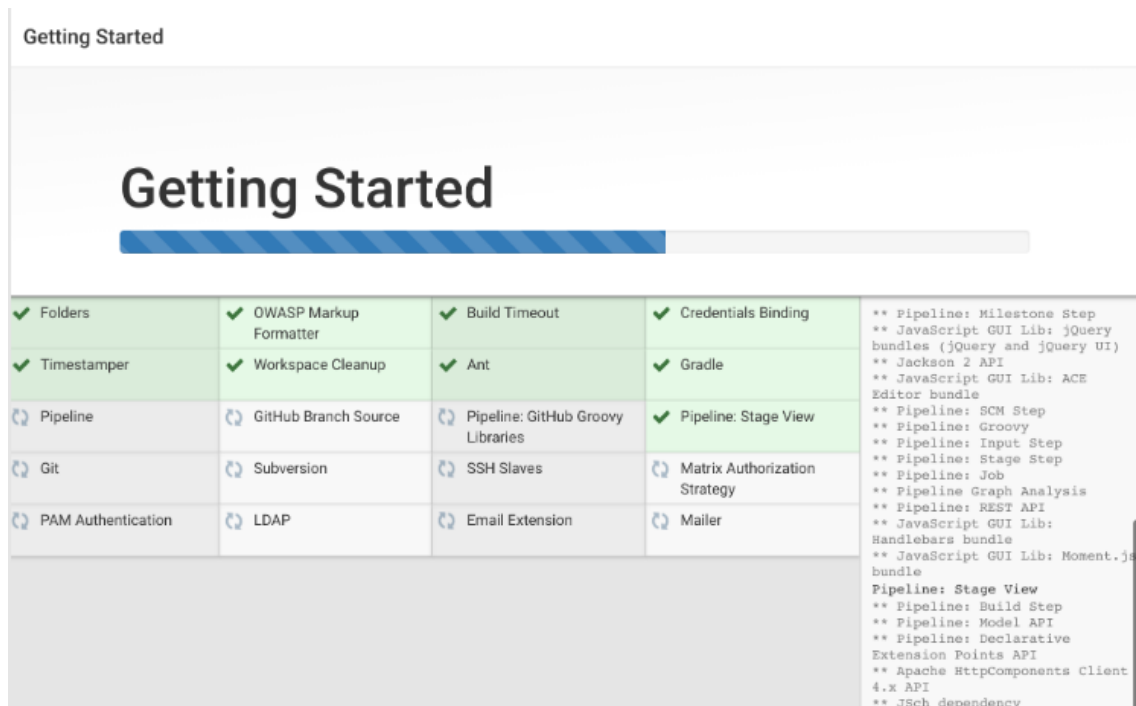
```
sudo cat /var/lib/jenkins/secrets/initialAdminPassword
```

```
capgemini@jenkins:~$ sudo cat /var/lib/jenkins/secrets/initialAdminPassword
faf2a518040f46f7be5bce2ea333fd12
```

Now you have successfully accessed in. You can click "Install suggested plugins" to install recommended plugins stated by Jenkins



After clicking “Install suggested plugins”, it’ll immediately begin the installation process



When the installation has completed, you'll be prompted to give Admin User Credentials.

Getting Started

Create First Admin User

Username:

Password:

Confirm password:

Full name:

E-mail address:

Jenkins 2.121.1

[Continue as admin](#)

Enter your Username and Password and then click “Save and Continue”. You will see an Instance Configuration page that will ask you to confirm the preferred URL for your Jenkins instance. Confirm either the domain name for your server or your server’s IP address:

Getting Started

Instance Configuration

Jenkins URL:

The Jenkins URL is used to provide the root URL for absolute links to various Jenkins resources. That means this value is required for proper operation of many Jenkins features including email notifications, PR status updates, and the `BUILD_URL` environment variable provided to build steps.

The proposed default value shown is not saved yet and is generated from the current request, if possible. The best practice is to set this value to the URL that users are expected to use. This will avoid confusion when sharing or viewing links.

Jenkins 2.121.1

[Not now](#) [Save and Finish](#)

Now, you've successfully completed setting up Jenkins

Getting Started

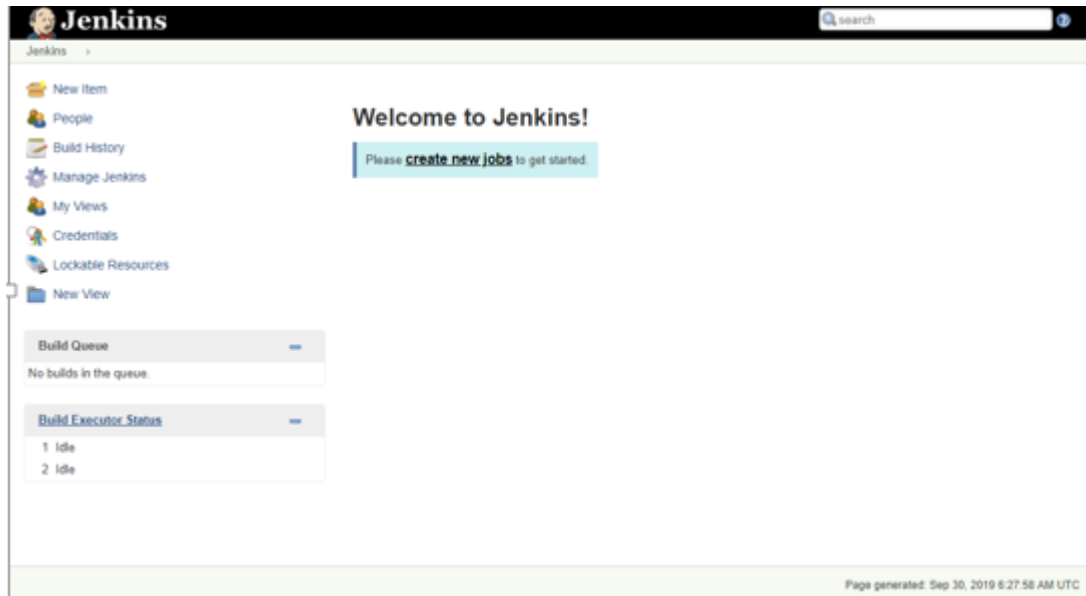
Jenkins is ready!

Your Jenkins setup is complete.

[Start using Jenkins](#)

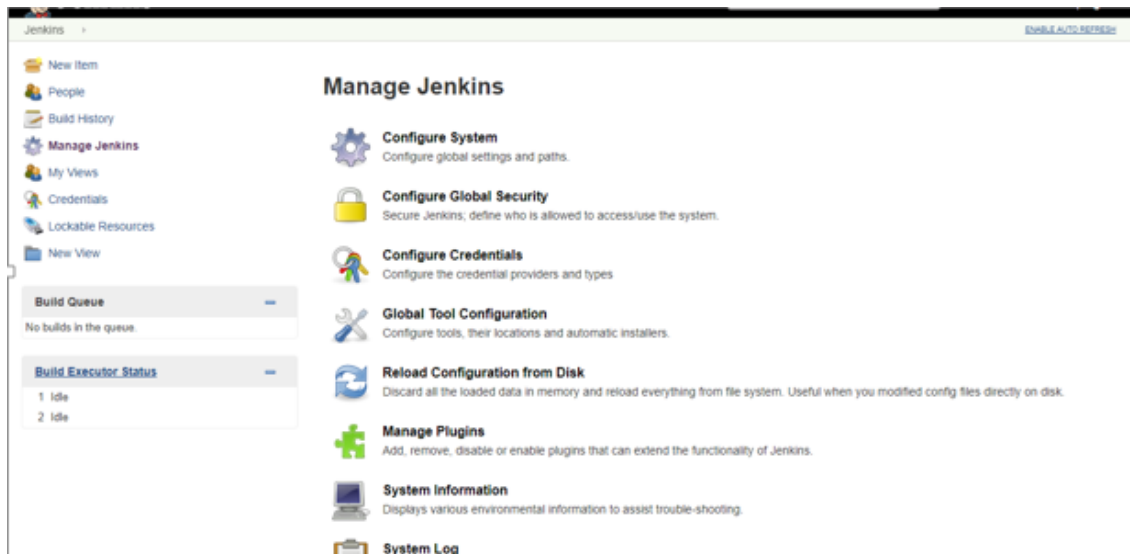
3 Working on Jenkins

This is the standard UI you'll get after you completed installing Jenkins and deploying it.



3.1 Install Maven in Jenkins server

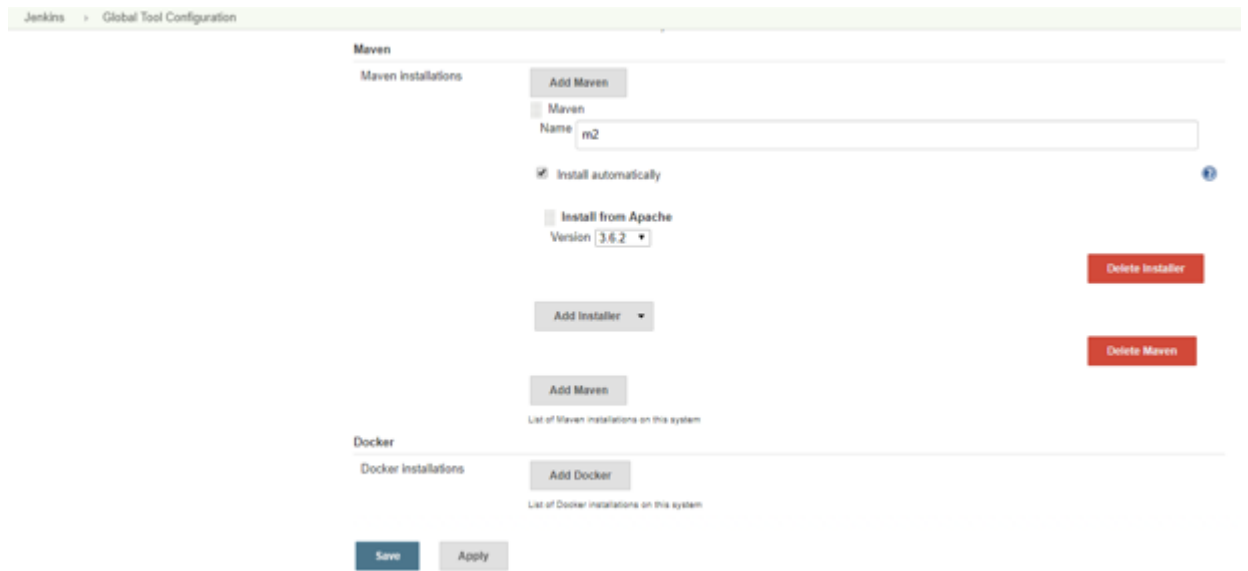
1. Go to Jenkins Home page,



On right side Click on Manage Jenkins

2. Click on Global Tool Configuration

In the Global Tool Configuration screen, scroll down till you see the Maven section and then click on the 'Add Maven' button.

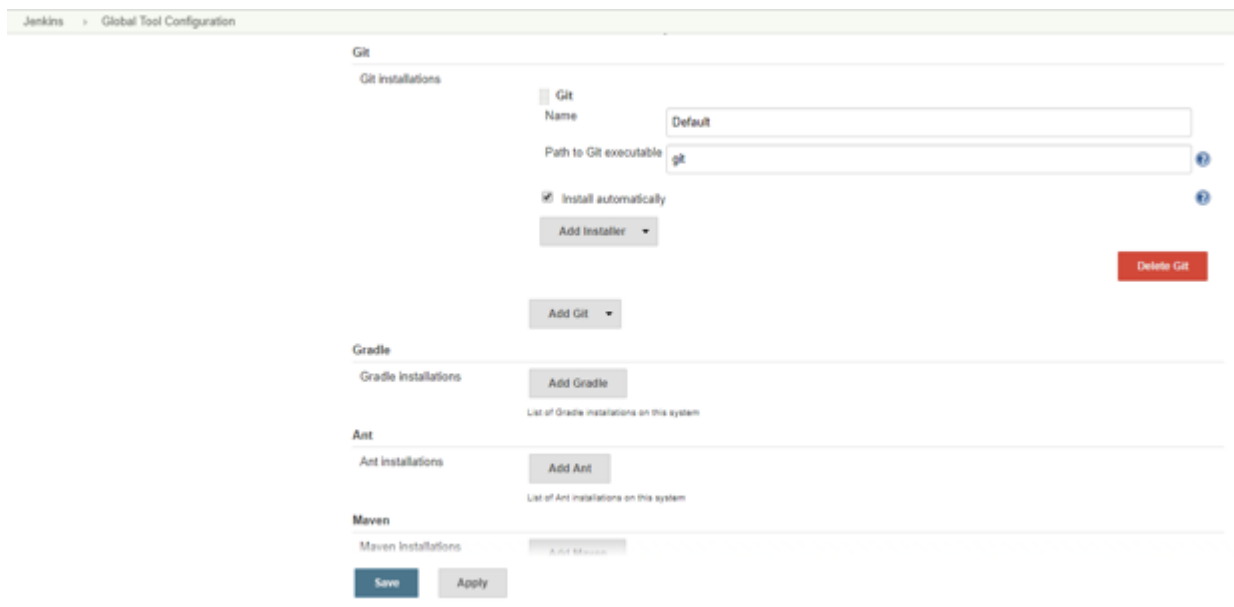


The screenshot shows the Jenkins 'Global Tool Configuration' page, specifically the 'Maven' section. The 'Maven Installations' list contains one entry named 'm2'. The configuration for 'm2' includes the 'Name' field set to 'm2', the 'Install automatically' checkbox checked, and the 'Install from Apache' section with the 'Version' dropdown set to '3.6.2'. There are buttons for 'Add Maven', 'Delete Installer', and 'Delete Maven'. Below the Maven section is the 'Docker' section with an 'Add Docker' button. At the bottom are 'Save' and 'Apply' buttons.

Enter the name of Maven to identify, Click on Install Automatically check-box and Select type of version according to the project specification and click apply.

3.2 Install Git in Jenkins server

1. In Global Tool Configuration scroll down to Git section and then click 'Add git' button
2. Give the path as git

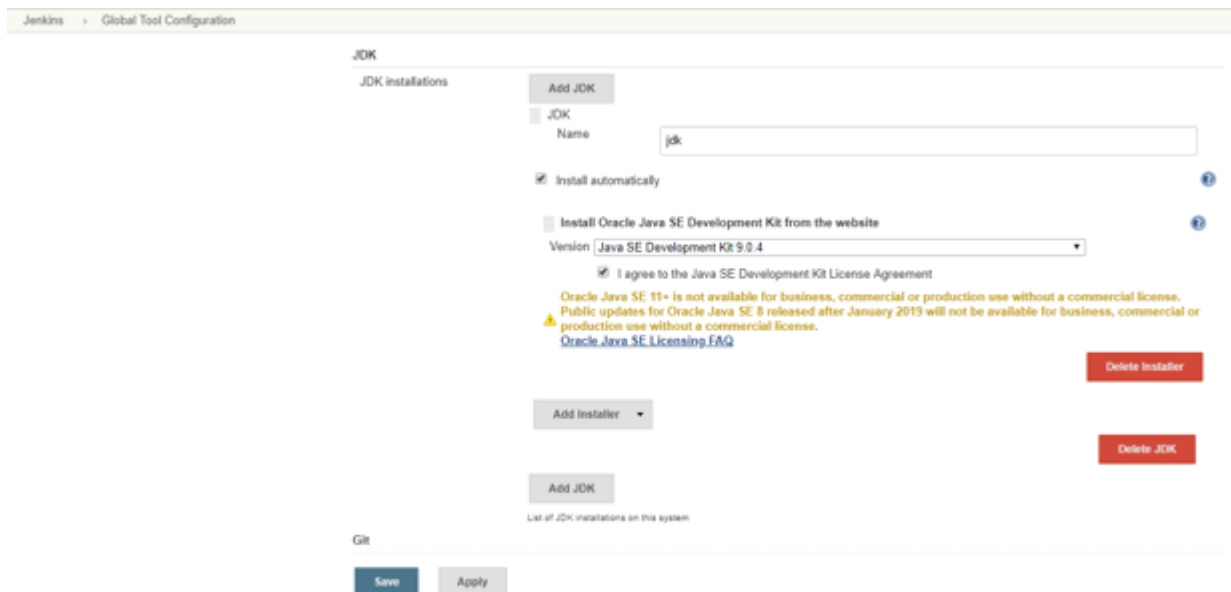


The screenshot shows the Jenkins 'Global Tool Configuration' page, specifically the 'Git' section. The 'Git Installations' list contains one entry named 'Default'. The configuration for 'Default' includes the 'Name' field set to 'Default', the 'Path to Git executable' field set to 'git', and the 'Install automatically' checkbox checked. There are buttons for 'Add Installer', 'Add Git', and 'Delete Git'. Below the Git section are sections for 'Gradle', 'Ant', and 'Maven', each with an 'Add' button. At the bottom are 'Save' and 'Apply' buttons.

3. Click on Install Automatically and apply

3.3 Install JDK in Jenkins server

1. In Global Tool Configuration scroll down to JDK section and then click 'Add jdk' button
2. Click on install Automatically and you will be asked oracle credentials, provide them and click on agreed.



The screenshot shows the Jenkins 'Global Tool Configuration' page for 'JDK'. The 'JDK installations' section is active. It features an 'Add JDK' button at the top. Below it, a 'JDK' checkbox is checked, and a 'Name' field contains the text 'jdk'. The 'Install automatically' checkbox is also checked. Underneath, there is a section for 'Install Oracle Java SE Development Kit from the website', with a 'Version' dropdown menu set to 'Java SE Development Kit 9.0.4'. A checkbox for 'I agree to the Java SE Development Kit License Agreement' is checked. A warning message states: 'Oracle Java SE 11+ is not available for business, commercial or production use without a commercial license. Public updates for Oracle Java SE 8 released after January 2019 will not be available for business, commercial or production use without a commercial license.' A link to 'Oracle Java SE Licensing FAQ' is provided. On the right side, there are 'Delete Installer' and 'Delete JDK' buttons. At the bottom, there is an 'Add Installer' dropdown, another 'Add JDK' button, and a 'List of JDK installations on this system' section. The 'Git' section is partially visible at the bottom, with 'Save' and 'Apply' buttons.

3. Click on apply and save button

3.4 Installing required plugin

1. Installing the required plugin in Jenkins dashboard
 - a. On the Jenkins dashboard, click on **Manage Jenkins** and select **Manage Plugins**. Click on the **Available** tab and write github plugin in the search box.
 - b. Click the checkbox and click on the button, **Download now and install after restart**.
 - c. Restart Jenkins

4. Building Pipe-line

A Pipeline can be created in one of the following ways:

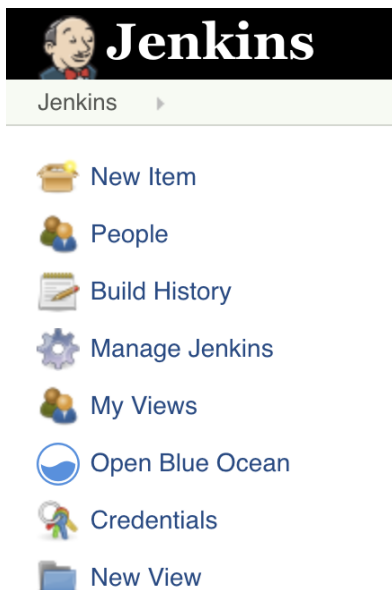
- **Through the classic UI** - you can enter a basic Pipeline directly in Jenkins through the classic UI.
- **In SCM** - you can write a Jenkinsfile manually, which you can commit to your project's source control repository.

4.1 Through the classic UI

A Jenkinsfile created using the classic UI is stored by Jenkins itself (within the Jenkins home directory).

To create a basic Pipeline through the Jenkins classic UI:

1. If required, ensure you are logged in to Jenkins.
2. From the Jenkins home page (i.e. the Dashboard of the Jenkins classic UI), click **New Item** at the top left.



3. In the **Enter an item name** field, specify the name for your new Pipeline project.
Caution: Jenkins uses this item name to create directories on disk. It is recommended to avoid using spaces in item names, since doing so may uncover bugs in scripts that do not properly handle spaces in directory paths.

4. Scroll down and click **Pipeline**, then click **OK** at the end of the page to open the Pipeline configuration page (whose **General** tab is selected).



5. Click the **Pipeline** tab at the top of the page to scroll down to the **Pipeline** section.
Note: If instead you are defining your Jenkinsfile in source control, follow the instructions in In SCM below.
6. In the **Pipeline** section, ensure that the **Definition** field indicates the **Pipeline script** option.
7. Enter your Pipeline code into the **Script** text area. For instance, copy the following Declarative example Pipeline code (below the *Jenkinsfile* heading) or its Scripted version equivalent and paste this into the **Script** text area. (The Declarative example below is used throughout the remainder of this procedure.)

Jenkinsfile (Declarative Pipeline)

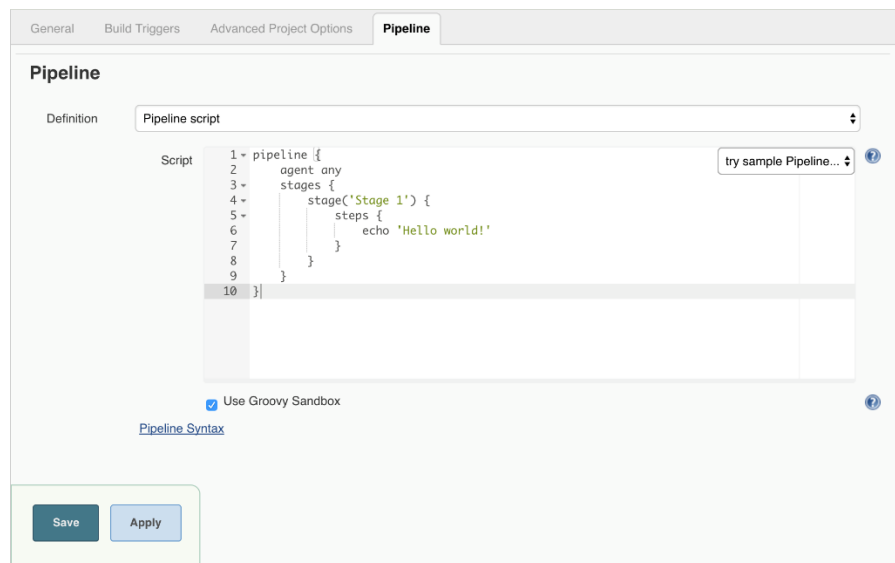
```
pipeline {  
  agent any  
  stages {  
    stage('Stage 1') {  
      steps {
```

```

        echo 'Hello world!'
    }
}
}
}
}

```

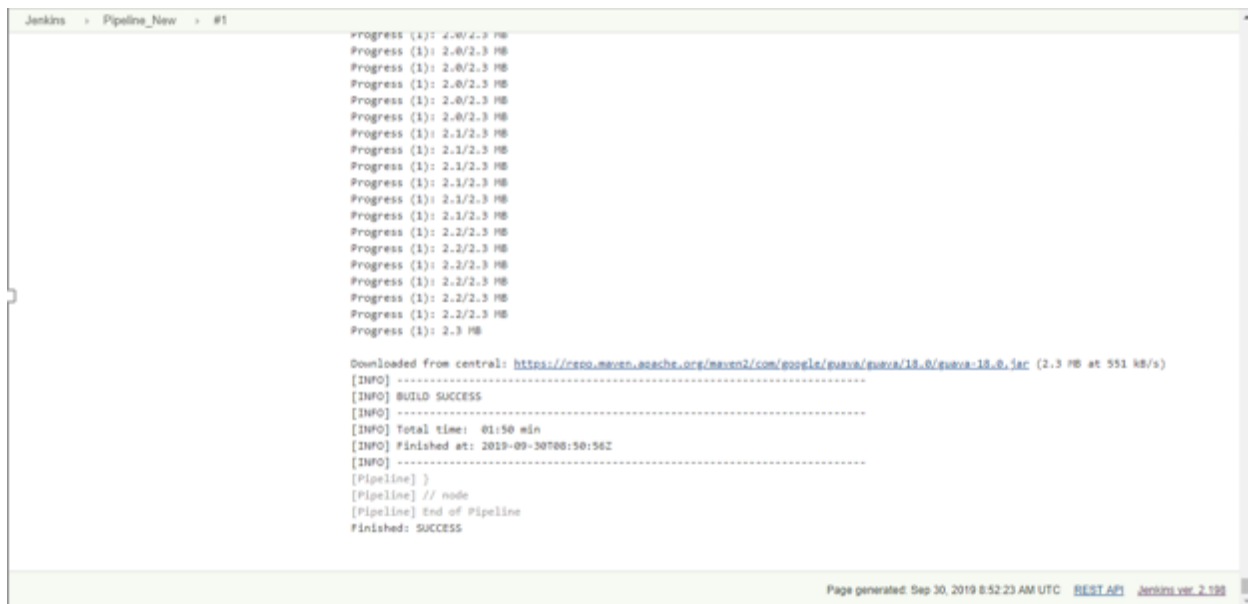
- a. agent instructs Jenkins to allocate an executor (on any available agent/node in the Jenkins environment) and workspace for the entire Pipeline
- b. echo writes simple string in the console output.
- c. node effectively does the same as agent (above).



Note: You can also select from canned *Scripted* Pipeline examples from the **try sample Pipeline** option at the top right of the **Script** text area. Be aware that there are no canned Declarative Pipeline examples available from this field.

8. Click **Save** to open the Pipeline project/item view page.
9. If you want to deploy a spring-boot application, first keep the application in github and then go to your pipeline and configure.
10. In general select GitHub project and provide your git application url.
11. Then click on save.
12. On this page, click **Build Now** on the left to run the Pipeline.

13. Under **Build History** on the left, click **#1** to access the details for this particular Pipeline run.
14. Click **Console Output** to see the full output from the Pipeline run. The following output shows a successful run of your Pipeline.



The screenshot shows the Jenkins web interface for a pipeline named 'Pipeline_New' at run #1. The console output is displayed in a monospaced font. It begins with a series of 'Progress' messages indicating the download of various Maven artifacts, such as 'Progress (1): 4.0/4.0 MB' and 'Progress (1): 2.0/2.3 MB'. This is followed by a message 'Downloaded from central: https://repo.maven.apache.org/maven2/com/google/guava/guava/18.0/guava-18.0.jar (2.3 MB at 551 kB/s)'. The output then shows '[INFO] BUILD SUCCESS' and '[INFO] Total time: 01:50 min'. The final status is 'Finished: SUCCESS'. At the bottom right, a footer indicates 'Page generated: Sep 30, 2019 8:52:23 AM UTC', a 'REST API' link, and 'Jenkins ver. 2.198'.

```
Jenkins > Pipeline_New > #1

Progress (1): 4.0/4.0 MB
Progress (1): 2.0/2.3 MB
Progress (1): 2.0/2.3 MB
Progress (1): 2.0/2.3 MB
Progress (1): 2.0/2.3 MB
Progress (1): 2.0/2.3 MB
Progress (1): 2.1/2.3 MB
Progress (1): 2.1/2.3 MB
Progress (1): 2.1/2.3 MB
Progress (1): 2.1/2.3 MB
Progress (1): 2.1/2.3 MB
Progress (1): 2.2/2.3 MB
Progress (1): 2.2/2.3 MB
Progress (1): 2.2/2.3 MB
Progress (1): 2.2/2.3 MB
Progress (1): 2.2/2.3 MB
Progress (1): 2.2/2.3 MB
Progress (1): 2.3 MB

Downloaded from central: https://repo.maven.apache.org/maven2/com/google/guava/guava/18.0/guava-18.0.jar (2.3 MB at 551 kB/s)
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 01:50 min
[INFO] Finished at: 2019-09-30T08:50:56Z
[INFO] -----
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS

Page generated: Sep 30, 2019 8:52:23 AM UTC  REST API  Jenkins ver. 2.198
```

15. The pipeline is created and application is deployed in it.

5. Docker

Docker is a set of platform-as-a-service products that use OS-level virtualization to deliver software in packages called containers. Containers are isolated from one another and bundle their own software, libraries and configuration files; they can communicate with each other through well-defined channels.

5.1 Steps to install Docker

1. To install the docker use following command

```
sudo snap install -y docker
```

2. Create the docker group.

```
sudo groupadd docker
```

3. Next, add the **USER** to the **docker** group so you can execute Docker commands without using **sudo**. Note that you'll have to log out and log back in for the settings to take effect

```
sudo usermod -a -G docker $USER
```

4. Restart Your VM

5. Now to confirm to install, use the following command:

```
docker info
```

5.2 Building Docker Image

Docker containers are instantiated through docker images which are read only templates. Generally, we use images that are already present in the repository. But, for particular projects or applications we need to create new docker images through which we can get our required type of containers. To create our own docker image use the following commands:

Initially, create a new file directory:

```
mkdir dockerprojects
```

```
capgemini@jenkins:~$ mkdir dockerprojects
```

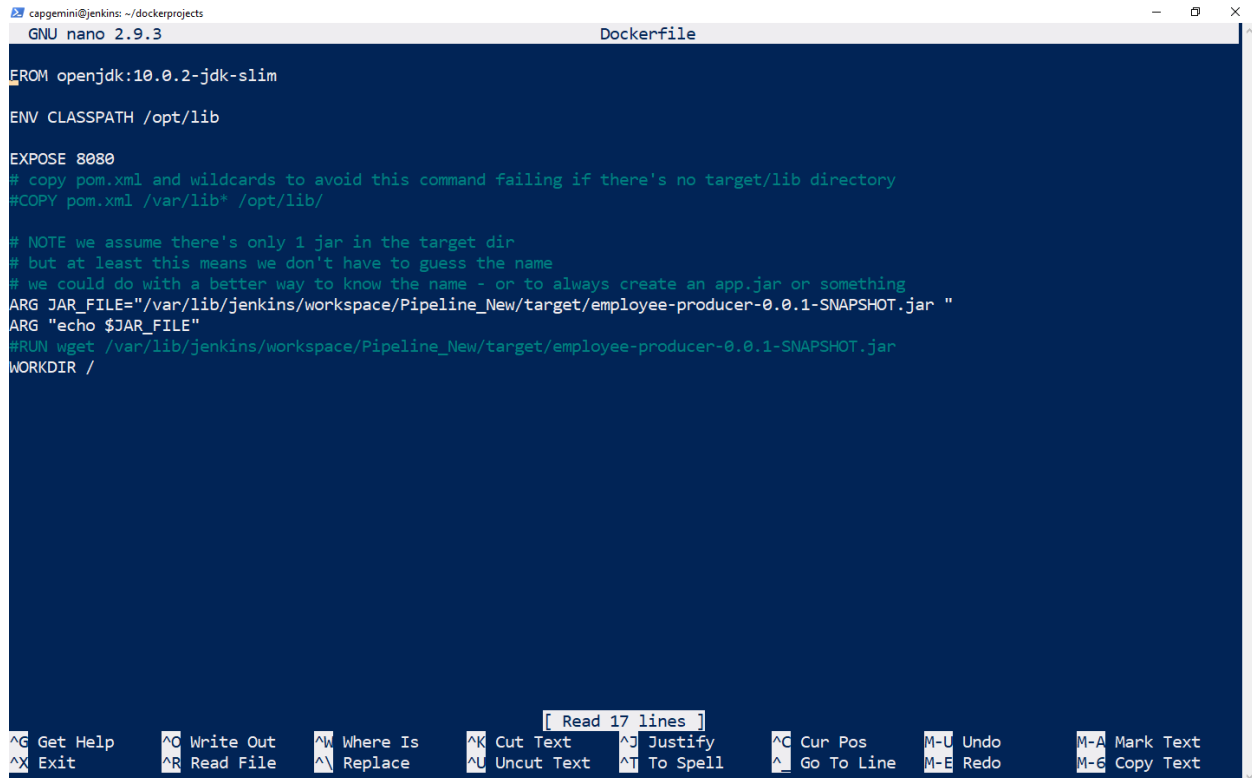
Navigate into the created directory using `cd dockerprojects`

Create a new file with Dockerfile as its name:

nano Dockerfile

```
capgemini@jenkins:~/dockerprojects$ nano Dockerfile
```

In the Dockerfile

A screenshot of a terminal window showing the nano text editor editing a Dockerfile. The window title is "capgemini@jenkins: ~/dockerprojects". The editor's status bar at the top shows "GNU nano 2.9.3" and "Dockerfile". The Dockerfile content includes: FROM openjdk:10.0.2-jdk-slim, ENV CLASSPATH /opt/lib, EXPOSE 8080, a comment about copying pom.xml, #COPY pom.xml /var/lib* /opt/lib/, a note about the target directory, ARG JAR_FILE="/var/lib/jenkins/workspace/Pipeline_New/target/employee-producer-0.0.1-SNAPSHOT.jar", ARG "echo \$JAR_FILE", #RUN wget /var/lib/jenkins/workspace/Pipeline_New/target/employee-producer-0.0.1-SNAPSHOT.jar, and WORKDIR /. The bottom status bar shows various nano editor shortcuts like ^G Get Help, ^O Write Out, ^W Where Is, ^K Cut Text, ^J Justify, ^C Cur Pos, M-U Undo, M-A Mark Text, ^X Exit, ^R Read File, ^_ Replace, ^U Uncut Text, ^T To Spell, ^_ Go To Line, M-E Redo, and M-G Copy Text. A small tooltip "Read 17 lines" is visible above the status bar.

Here, employee-producer-0.0.1-SNAPSHOT.jar is created from simple spring boot application. We are giving this jar file as JAR path for Jenkins to automate the execution.

Now create your new image and provide it with a name (run these commands within the same directory):

```
docker build -t {image-name} .
```

I've specified the docker image name as jenkins-image.

Now run it by using following command:

```
docker run --name {jenkins-name} -it -d {jenkins-image}
```

You'll be getting some random letters and numbers as output. It is an UUID long identifier. You can operate that container using that long UUID.

Now, check in docker images using the command

```
docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
jenkins-image	latest	4741a1a1f046	2 minutes ago	498MB
openjdk	10.0.2-jdk-slim	02f0ec2751d2	10 months ago	498MB

Check if the container is running by using following command:

```
docker ps
```

CONTAINER ID	IMAGE NAMES	COMMAND	CREATED	STATUS
3c28bf7eeecb	jenkins-image	"jshell"	About a minute ago	Up About a minute
8080/tcp	jenkins-name			

Now, the jar file of application that is deployed in Jenkins is placed in a container and image in docker

6. Kubernetes on Azure

6.1 Kubernetes Introduction

Kubernetes is an open-source container-orchestration system for automating application deployment, scaling, and management. It was originally designed by Google, and is now maintained by the Cloud Native Computing Foundation.

Containers are a good way to bundle and run your applications. In a production environment, you need to manage the containers that run the applications and ensure that there is no downtime. For example, if a container goes down, another container needs to start. Wouldn't it be easier if this behavior was handled by a system?

That's how Kubernetes comes to the rescue! Kubernetes provides you with a framework to run distributed systems resiliently. It takes care of your scaling requirements, failover, deployment patterns, and more. For example, Kubernetes can easily manage a canary deployment for your system.

- **Service discovery and load balancing**

Kubernetes can expose a container using the DNS name or using their own IP address. If traffic to a container is high, Kubernetes is able to load balance and distribute the network traffic so that the deployment is stable.

- **Storage orchestration**

Kubernetes allows you to automatically mount a storage system of your choice, such as local storages, public cloud providers, and more.

- **Automated rollouts and rollbacks**

You can describe the desired state for your deployed containers using Kubernetes, and it can change the actual state to the desired state at a controlled rate. For example, you can automate Kubernetes to create new containers for your deployment, remove existing containers and adopt all their resources to the new container.

- **Automatic bin packing**

Kubernetes allows you to specify how much CPU and memory (RAM) each container needs. When containers have resource requests specified, Kubernetes can make better decisions to manage the resources for containers.

- **Self-healing**

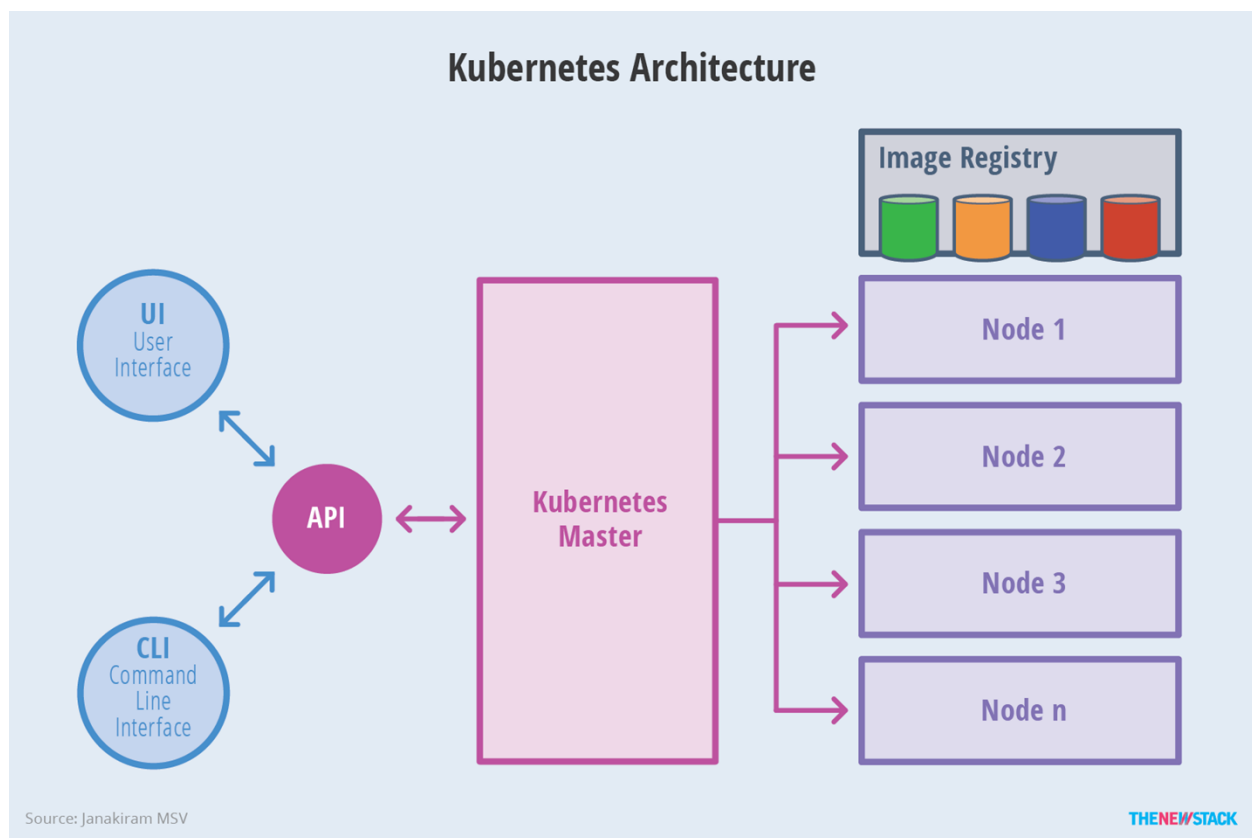
Kubernetes restarts containers that fail, replaces containers, kills containers that don't

respond to your user-defined health check, and doesn't advertise them to clients until they are ready to serve.

- **Secret and configuration management**

Kubernetes lets you store and manage sensitive information, such as passwords, OAuth tokens, and ssh keys. You can deploy and update secrets and application configuration without rebuilding your container images, and without exposing secrets in your stack configuration.

6.1.1Kubernetes Architecture



Kubernetes - Master Machine Components

Following are the components of Kubernetes Master Machine.

etcd

It stores the configuration information which can be used by each of the nodes in the cluster. It is a high availability key value store that can be distributed among multiple nodes. It is accessible only by Kubernetes API server as it may have some sensitive information. It is a distributed key value Store which is accessible to all.

API Server

Kubernetes is an API server which provides all the operation on cluster using the API. API server implements an interface, which means different tools and libraries can readily communicate with it. Kubeconfig is a package along with the server side tools that can be used for communication. It exposes Kubernetes API.

Controller Manager

This component is responsible for most of the controllers that regulates the state of cluster and performs a task. In general, it can be considered as a daemon which runs in nonterminating loop and is responsible for collecting and sending information to API server. It works toward getting the shared state of cluster and then make changes to bring the current status of the server to the desired state. The key controllers are replication controller, endpoint controller, namespace controller, and service account controller. The controller manager runs different kind of controllers to handle nodes, endpoints, etc.

Scheduler

This is one of the key components of Kubernetes master. It is a service in master responsible for distributing the workload. It is responsible for tracking utilization of working load on cluster nodes and then placing the workload on which resources are

available and accept the workload. In other words, this is the mechanism responsible for allocating pods to available nodes. The scheduler is responsible for workload utilization and allocating pod to new node.

Kubernetes - Node Components

Following are the key components of Node server which are necessary to communicate with Kubernetes master.

Docker

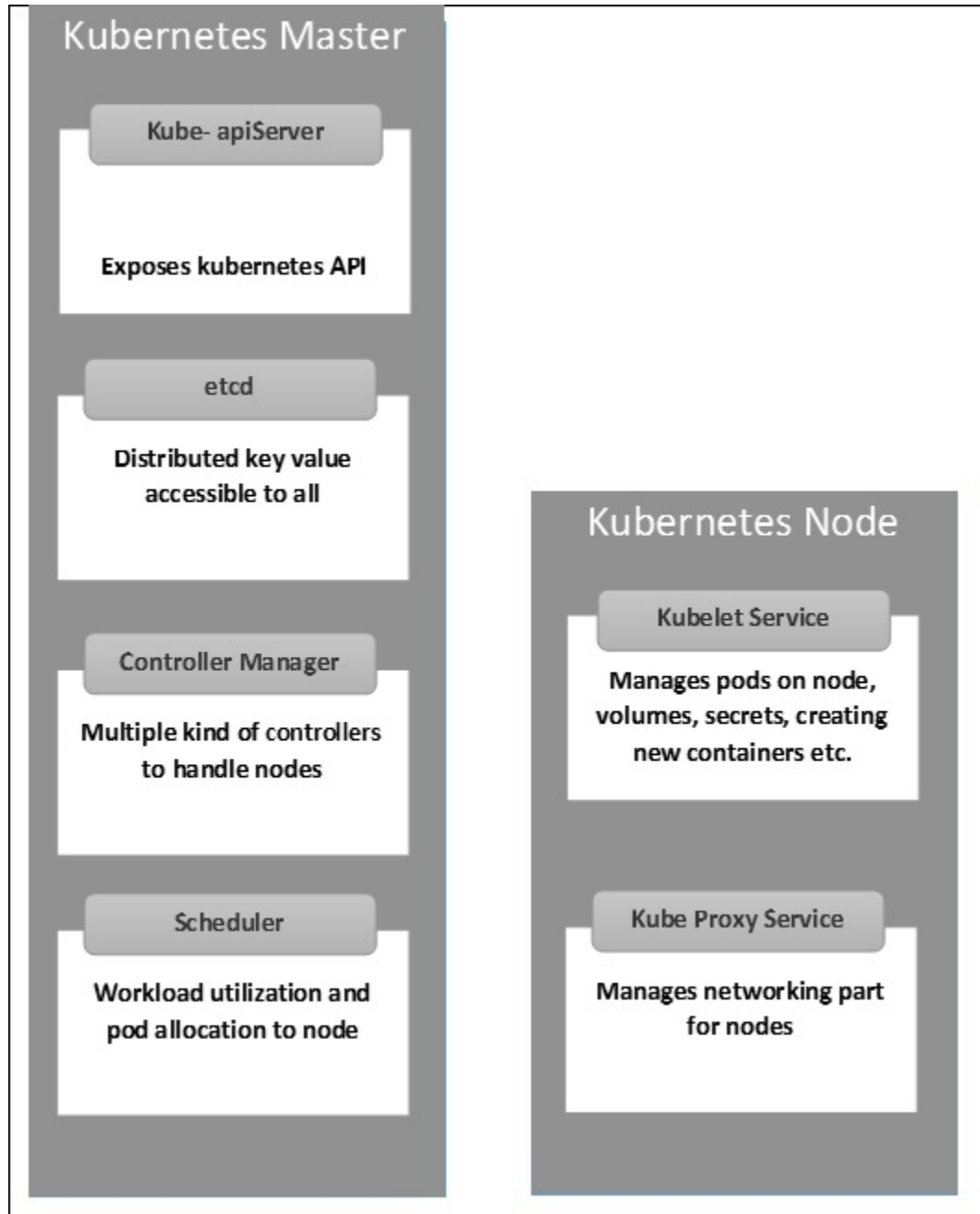
The first requirement of each node is Docker which helps in running the encapsulated application containers in a relatively isolated but lightweight operating environment.

Kubelet Service

This is a small service in each node responsible for relaying information to and from control plane service. It interacts with etcd store to read configuration details and write values. This communicates with the master component to receive commands and work. The kubelet process then assumes responsibility for maintaining the state of work and the node server. It manages network rules, port forwarding, etc.

Kubernetes Proxy Service

This is a proxy service which runs on each node and helps in making services available to the external host. It helps in forwarding the request to correct containers and is capable of performing primitive load balancing. It makes sure that the networking environment is predictable and accessible and at the same time it is isolated as well. It manages pods on node, volumes, secrets, creating new containers' health checkup, etc.



You can find the detailed guide on Kubernetes on the following link below:

<https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/>

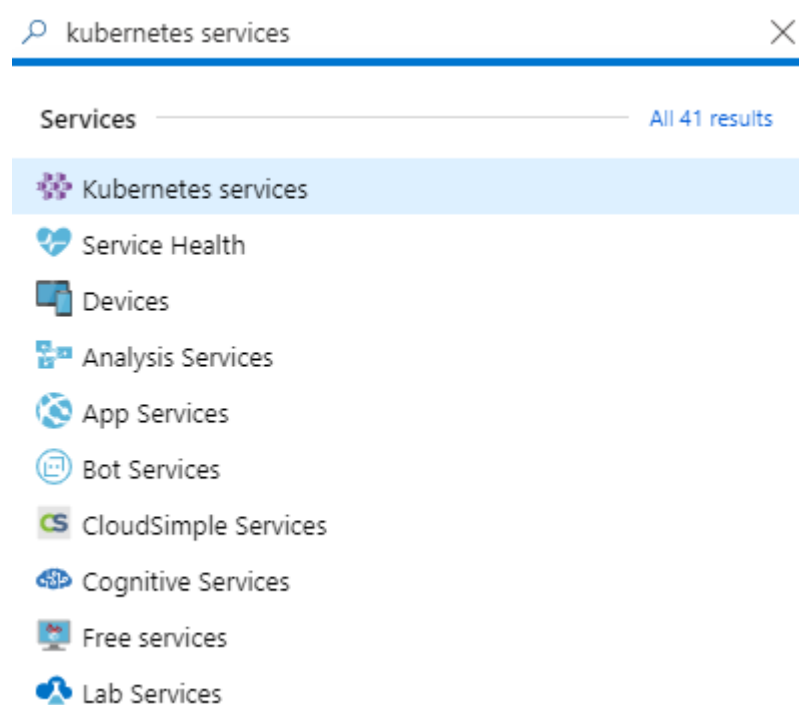
There are a lot of ways to install Kubernetes, for example, using minikube, using “EKS”, using “AKS” etc. In this documentation we will be using AKS (Azure Kubernetes Service)

Azure Kubernetes Service (AKS) is a managed container orchestration service, based on the open source Kubernetes system, which is available on the Microsoft Azure public cloud. An organization can use AKS to deploy, scale and manage Docker containers and container-based applications across a cluster of container hosts.

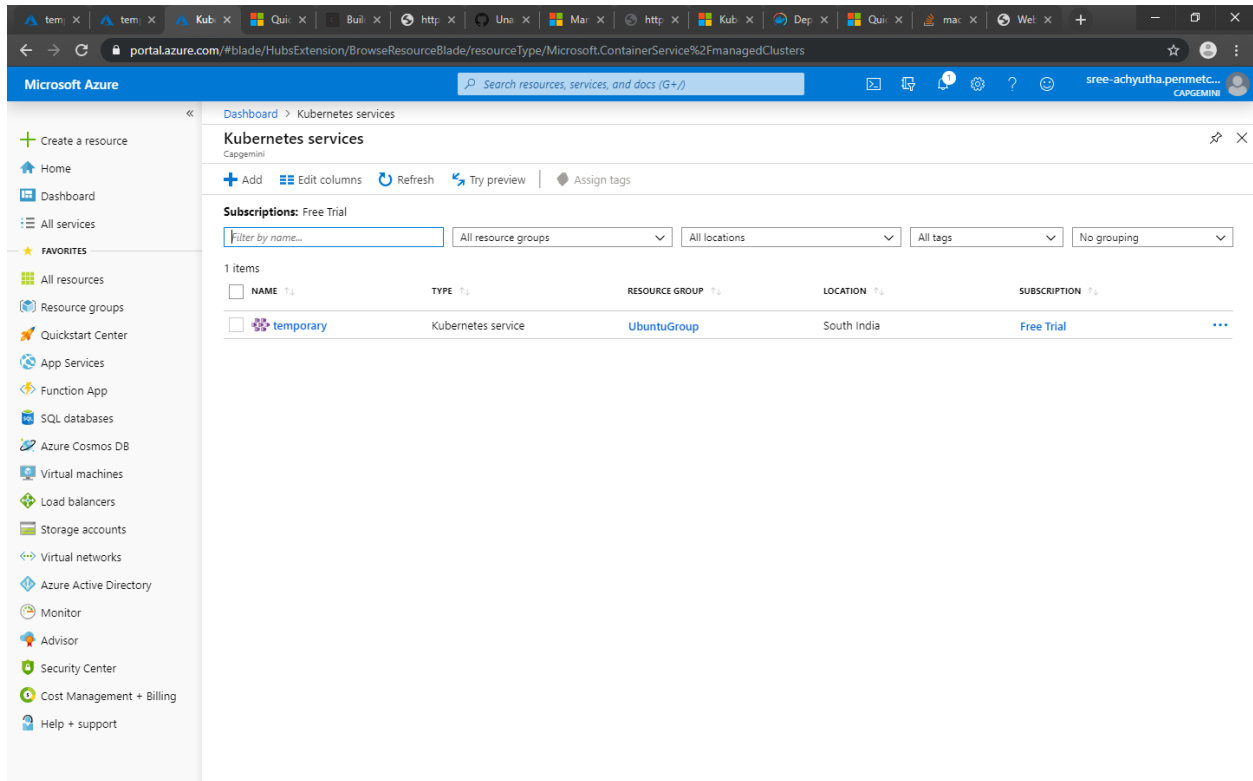
AKS is basically free to use, but you should pay for the resources that AKS have used such as storage, computing power, using VMs etc.

6.2 Instantiating Kubernetes Service in Azure Portal

You can deploy kubernetes application in Azure Portal. Just search for “kubernetes services” in search bar in Azure portal.



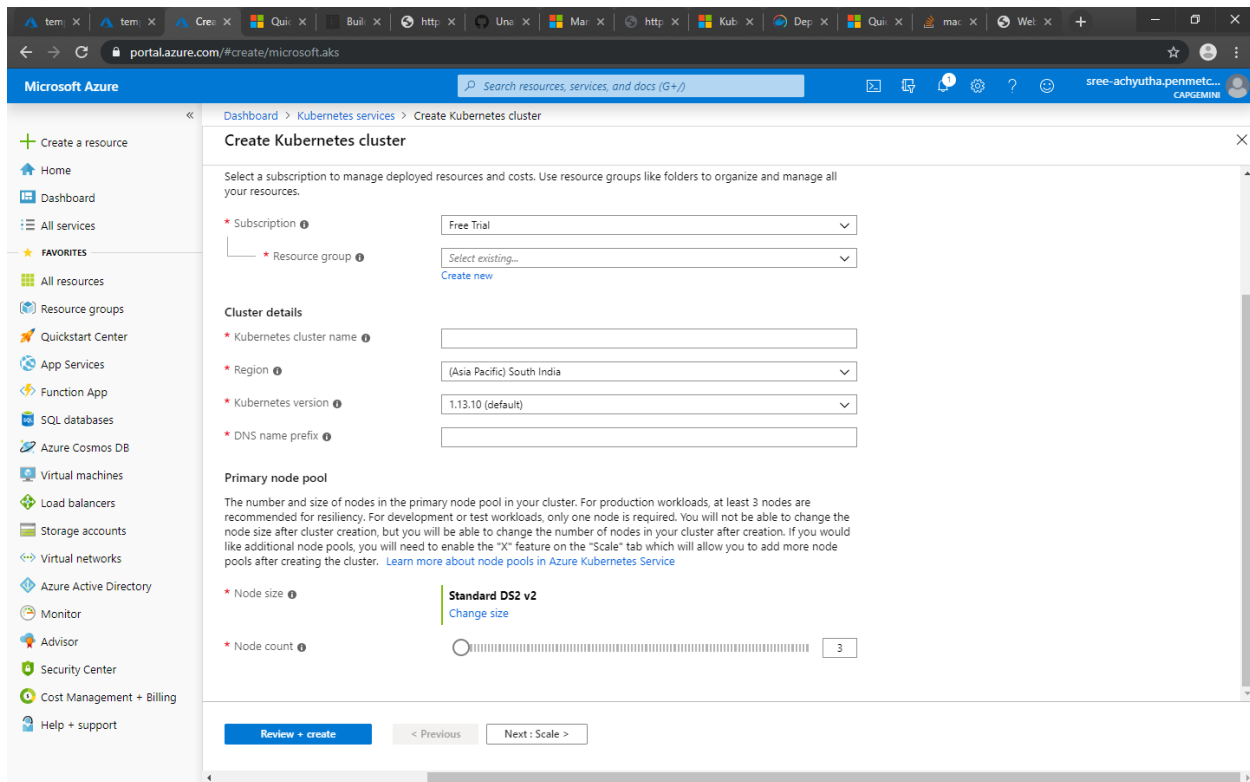
You'll get this Kubernetes Services Dashboard. Here I've already created the Kubernetes Cluster named as "temporary" in "UbuntuGroup" resource group



The screenshot shows the Microsoft Azure portal interface. The left sidebar contains navigation links for 'Create a resource', 'Home', 'Dashboard', 'All services', and 'FAVORITES'. The main content area is titled 'Kubernetes services' and includes a '+ Add' button, 'Edit columns', 'Refresh', 'Try preview', and 'Assign tags' options. Below this, there are filters for 'Subscriptions: Free Trial', 'Filter by name...', 'All resource groups', 'All locations', 'All tags', and 'No grouping'. A table lists 1 item with the following details:

NAME	TYPE	RESOURCE GROUP	LOCATION	SUBSCRIPTION
temporary	Kubernetes service	UbuntuGroup	South India	Free Trial

You can click on "+Add" to make your own Kubernetes Cluster. You can specify your own cluster name, your resource group and choose the Node Count size.



After giving necessary information, Click on “Review + create” to instantiate the kubernetes application. It’ll take some time to get started. Until then, you need to install Azure cli on your Ubuntu VM.

6.3 Installing Azure CLI

In order to install Azure CLI, you can use the following command below.

```
curl -sL https://aka.ms/InstallAzureCLIDeb | sudo bash
```

Note that you have to use Ubuntu 16.04+ and Debian 8+. If you are using other Distributions of Linux, go to the link below on how to install it manually.

<https://docs.microsoft.com/en-us/cli/azure/install-azure-cli-apt?view=azure-cli-latest>

```
Preparing to unpack .../azure-cli_2.0.74-1~bionic_all.deb ...
Unpacking azure-cli (2.0.74-1~bionic) ...
Setting up azure-cli (2.0.74-1~bionic) ...
capgemini@jenkins:~$ az
```


Once if you've done installing Azure CLI, just type "az" to get any output

```
cagmini@jenkins:~$ az
```

```
Welcome to Azure CLI!
-----
Use `az -h` to see available commands or go to https://aka.ms/cli.
```

```
Telemetry
-----
The Azure CLI collects usage data in order to improve your experience.
The data is anonymous and does not include commandline argument values.
The data is collected by Microsoft.
```

```
You can change your telemetry settings with `az configure`.
```



```
Welcome to the cool new Azure CLI!
```

```
Use `az --version` to display the current version.
Here are the base commands:
```

account	: Manage Azure subscription information.
---------	--

Now you've successfully installed Azure cli, you have to login.

az login

It'll provide you the link and passcode, login to that link using the passcode provided and give your azure credentials. You'll get JSON output once you've completed verifying the account.

```
capgemini@jenkins: $ az login
To sign in, use a web browser to open the page https://microsoft.com/devicelogin and enter the code HPCBC9HDB to authenticate.
[
{
  "cloudName": "AzureCloud",
  "id": "59310af4-3fb5-421b-af40-79bba05fc477",
  "isDefault": true,
  "name": "Free Trial",
  "state": "Enabled",
  "tenantId": "76a2ae5a-9f00-4f6b-95ed-5d33d77c4d61",
  "user": {
    "name": "sree-achyutha.penmetcha@capgemini.com",
    "type": "user"
  }
}
]
```

6.4 Adding Kubernetes Services to Ubuntu VM

Since you've created the kubernetes service in Azure portal, we need to link it to Ubuntu VM. I've instantiated the Kubernetes service named as "temporary" in resource group "UbuntuGroup". I'll be merging that to Ubuntu VM using the command below:

```
az aks get-credentials --resource-group UbuntuGroup --name temporary
```

```
capgemini@jenkins:~$ az aks get-credentials --resource-group UbuntuGroup --name temporary
Merged "temporary" as current context in /home/capgemini/.kube/config
```

Now, you need to install Kubectl in Ubuntu. The Kubernetes command-line tool, kubectl, allows you to run commands against Kubernetes clusters. You can use kubectl to deploy applications, inspect and manage cluster resources, and view logs.

```
az aks install-cli
```

```
capgemini@jenkins:~$ sudo az aks install-cli
Downloading client to "/usr/local/bin/kubectl" from "https://storage.googleapis.com/kubernetes-release/release/v1.16.1/bin/linux/amd64/kubectl"
Please ensure that /usr/local/bin is in your search PATH, so the `kubectl` command can be found.
```

You need to add /usr/local/bin to search path such that Ubuntu can detect "kubectl" command. In order to do that, use the command below:

```
export PATH=/usr/local/git/bin:/usr/local/bin:$PATH
```

Now verify that kubectl has been installed. Use "kubectl version" to display it's version.

```
capgemini@jenkins:~$ kubectl version
Client Version: version.Info{Major:"1", Minor:"16", GitVersion:"v1.16.1", GitCommit:"d647ddbd755faf07169599a625faf302ffc34458", GitTreeState:"clean", BuildDate:"2019-10-02T17:01:15Z", GoVersion:"go1.12.10", Compiler:"gc", Platform:"linux/amd64"}
Server Version: version.Info{Major:"1", Minor:"13", GitVersion:"v1.13.10", GitCommit:"37d169313237cb4ceb2cc4bef300f2ae3053c1a2", GitTreeState:"clean", BuildDate:"2019-08-19T10:44:49Z", GoVersion:"go1.11.13", Compiler:"gc", Platform:"linux/amd64"}
```

When it's completed installing, you can use the command below to get kubernetes nodes:

```
kubectl get nodes
```

```
capgemini@jenkins:~$ kubectl get nodes
```

NAME	STATUS	ROLES	AGE	VERSION
aks-agentpool-30326481-0	Ready	agent	88m	v1.13.10

You can even verify the nodes on Virtual Machines Dashboard.

The screenshot shows the Microsoft Azure portal interface. The left sidebar contains navigation links for various services. The main content area displays the 'Virtual machines' dashboard for the 'Capgemini' subscription. It includes a table with the following data:

NAME	TYPE	STATUS	RESOURCE GROUP	LOCATION	SOURCE	MAINTENANCE STAT...	SUBSCRIPTION
aks-agentpool-30326481-0	Virtual machine	Running	MC_UbuntuGroup...	South India	Marketplace	-	Free Trial
jenkins	Virtual machine	Running	UbuntuGroup	South India	Marketplace	-	Free Trial

6.5 Deploying Simple Azure-Voting Application on Kubernetes

Now, you can deploy simple software web application on Kubernetes cluster. You can use the link below for detailed guide on deploying simple application on Kubernetes Cluster using AKS.

<https://docs.microsoft.com/en-us/azure/aks/kubernetes-walkthrough-portal>

By using nano or vi, create azure-vote.yaml and copy the following.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: azure-vote-back
spec:
  replicas: 1
  selector:
    matchLabels:
      app: azure-vote-back
  template:
    metadata:
      labels:
        app: azure-vote-back
    spec:
      nodeSelector:
        "beta.kubernetes.io/os": linux
      containers:
        - name: azure-vote-back
          image: redis
          resources:
            requests:
              cpu: 100m
              memory: 128Mi
            limits:
              cpu: 250m
              memory: 256Mi
          ports:
            - containerPort: 6379
              name: redis
---
apiVersion: v1
kind: Service
metadata:
  name: azure-vote-back
spec:
  ports:
    - port: 6379
  selector:
    app: azure-vote-back
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: azure-vote-front
spec:
  replicas: 1
  selector:
    matchLabels:
      app: azure-vote-front
  template:
    metadata:
```

```

labels:
  app: azure-vote-front
spec:
  nodeSelector:
    "beta.kubernetes.io/os": linux
  containers:
  - name: azure-vote-front
    image: microsoft/azure-vote-front:v1
    resources:
      requests:
        cpu: 100m
        memory: 128Mi
      limits:
        cpu: 250m
        memory: 256Mi
    ports:
    - containerPort: 80
    env:
    - name: REDIS
      value: "azure-vote-back"
---
apiVersion: v1
kind: Service
metadata:
  name: azure-vote-front
spec:
  type: LoadBalancer
  ports:
  - port: 80
  selector:
    app: azure-vote-front

```

Now save that file. We'll be deploying this voting application using `kubectl apply` and “-f” represents file name.

```
kubectl apply -f azure-vote.yaml
```

```

capgemini@jenkins:~$ kubectl apply -f azure-vote.yaml
deployment.apps/azure-vote-back created
service/azure-vote-back created
deployment.apps/azure-vote-front created
service/azure-vote-front created

```

For testing the application, you can use the command below. To monitor progress, use the `kubectl get service` command with the `--watch` argument.

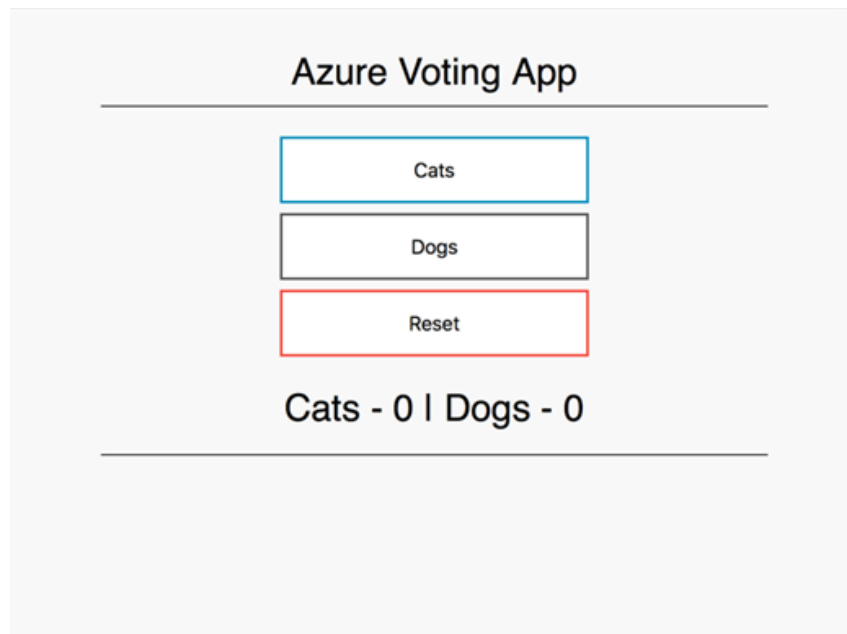
```
kubectl get service azure-vote-front --watch
```


When the application runs, a Kubernetes service exposes the application front end to the internet. This process can take a few minutes to complete. Use the External-IP to access this application from the browser.

```
capgemini@jenkins:~$ kubectl get service azure-vote-front --watch
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
azure-vote-front	LoadBalancer	10.0.42.102	52.172.38.50	80:32764/TCP	50m

Now you've successfully accessed a simple application on Kubernetes



6.6 Deploying Spring-Boot Application

Here is the link for the detailed guide on how to deploy spring your boot application.

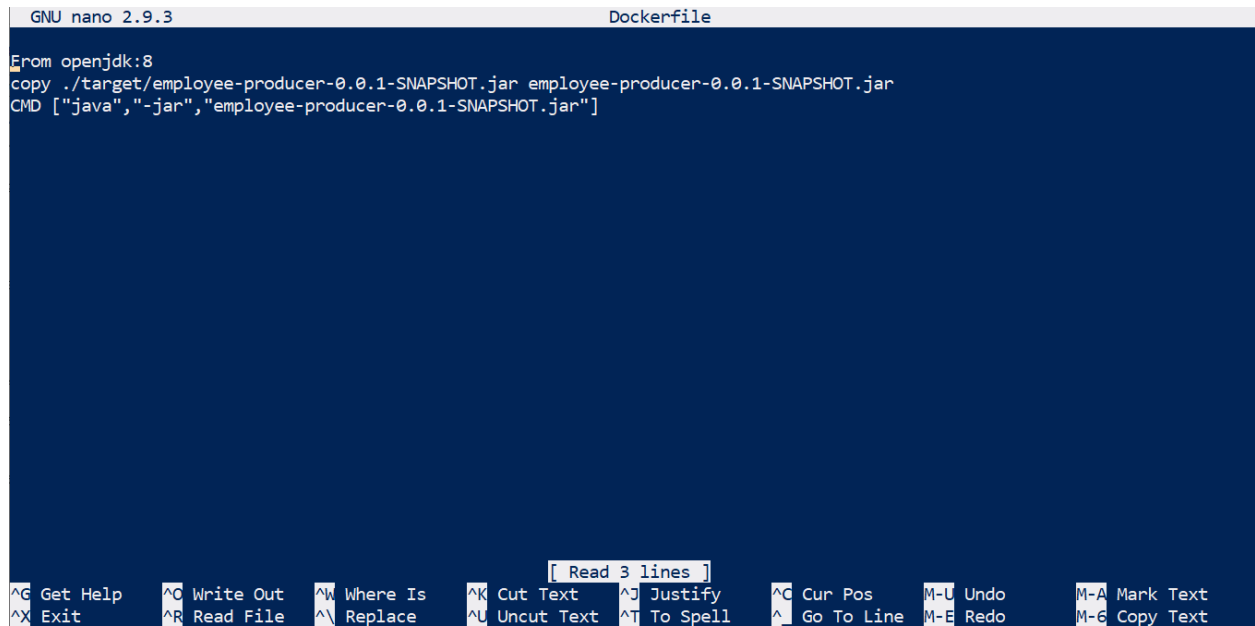
<https://gorillalogic.com/blog/build-and-deploy-a-spring-boot-app-on-kubernetes-minikube/>

Now, we are going to deploy a simple spring boot application container in Kubernetes. Make sure you've your Docker image ready.

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
producer	latest	6b34b7508766	5 seconds ago	502MB
openjdk	8	e8d00769c8a8	3 weeks ago	488MB

You can easily deploy your spring boot application on Docker image. All you need is Docker software, your spring-boot application and Dockerfile.

These are the contents I've wrote in Dockerfile.

A screenshot of a terminal window showing the nano text editor. The title bar at the top reads "GNU nano 2.9.3" on the left and "Dockerfile" on the right. The editor's content area has a dark blue background with white text. The text inside the editor is: "From openjdk:8", "copy ./target/employee-producer-0.0.1-SNAPSHOT.jar employee-producer-0.0.1-SNAPSHOT.jar", and "CMD [\"java\", \"-jar\", \"employee-producer-0.0.1-SNAPSHOT.jar\"]". At the bottom of the editor, there is a status bar with various keyboard shortcuts and their functions, such as "^G Get Help", "^O Write Out", "^W Where Is", etc. A small tooltip "[Read 3 lines]" is visible above the status bar.

After creating Dockerfile, type the following command:

```
docker image build -t producer .
```

```

achyuth@ubuntu:~/spring-2/employee-producer$ docker image build -t producer .
Sending build context to Docker daemon 40.91MB
Step 1/3 : From openjdk:8
8: Pulling from library/openjdk
092586df9206: Pull complete
ef599477fae0: Pull complete
4530c6472b5d: Pull complete
d34d61487075: Pull complete
272f46008219: Pull complete
12ff6ccfe7a6: Pull complete
f26b99e1adb1: Pull complete
Digest: sha256:350761f9310c2b6665ac5e62b15d03dce859bc20dff59d6dbc63b114d9c39001
Status: Downloaded newer image for openjdk:8
---> e8d00769c8a8
Step 2/3 : copy ./target/employee-producer-0.0.1-SNAPSHOT.jar employee-producer-0.0.1-SNAPSHOT.jar
---> aa9df63139a4
Step 3/3 : CMD ["java","-jar","employee-producer-0.0.1-SNAPSHOT.jar"]
---> Running in c72202d964ba
Removing intermediate container c72202d964ba
---> 6b34b7508766
Successfully built 6b34b7508766
Successfully tagged producer:latest

```

Now using this image, you can deploy it in kubernetes using kubectl. You can use the command below:

```
sudo kubectl run {DEPLOYMENT_NAME} --image= {YOUR_IMAGE} --port=8080
```

The command “kubectl run” only needs the {DEPLOYMENT_NAME} to work, but if you want to pull a Docker image inside this deployment, you should use the “--image” option, with which you can specify the Docker image to be used.

```

achyuth@ubuntu:~$ sudo kubectl run employee-producer --image=producer --port=8080
kubectl run --generator=deployment/apps.v1 is DEPRECATED and will be removed in a future version. Use kubectl run --generator=run-pod/v1 or kubectl create instead.
deployment.apps/employee-producer created

```

You can verify if it's been deployed by using the below command:

```
sudo kubectl get deployments
```

```

achyuth@ubuntu:~$ sudo kubectl get deployments

```

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
azure-vote-back	1/1	1	1	4m14s
azure-vote-front	0/1	1	0	4m14s
employee-producer	0/1	1	0	20s

Give some time for it to get deployed. After deploying it you should expose it to outside the cluster. You can do it by using this command:

```
sudo kubectl expose deployment/< Your deployment service name >--  
type="NodePort" --port=8080
```

```
achyuth@ubuntu:~/spring-2/employee-producer$ sudo kubectl expose deployment/employee-producer --type="NodePort" --port=8080  
service/employee-producer exposed
```

The logic behind the above command is the following: we want to expose our deployment to the world through the NodePort (which will be assigned when the service is created). After you execute the command, a console message will appear: “service ‘employee-producer’ exposed,” which means that the application can be accessed from outside the cluster. We now need to know the NodePort of the service that was created. To do this, in addition to obtaining more details about our service, we can use the following command:

```
kubectl describe services/<Your deployment service name>
```

```
^Cachyuth@ubuntu:~/spring-2/employee-producer$ kubectl describe services/employee-producer  
Name: employee-producer  
Namespace: default  
Labels: run=employee-producer  
Annotations: <none>  
Selector: run=employee-producer  
Type: NodePort  
IP: 10.0.82.9  
Port: <unset> 8080/TCP  
TargetPort: 8080/TCP  
NodePort: <unset> 30220/TCP  
Endpoints:  
Session Affinity: None  
External Traffic Policy: Cluster  
Events: <none>
```

Here you can see the nodeport that I got is “30220”. You can test your spring boot application on your browser by providing your IP address followed by 30220.

<http://<Your IP Address>:30220/employee>

```
{"empId": "1", "name": "emp1", "designation": "manager", "salary": 3000.0}
```