© DevOps

De

DeDe

**DevOps self intro:**

**Hi Good Morning !**

**My self manikanta, I’m having 4+ years of experience on devops and AWS**

**Past 3 years I have been working in XXXX organization as DevOps engineer**

**Well ,coming to my Qualification**

**I have completed my Graduation from Osmania University at Hyderbad.**

**Well, coming to my Skill set**

**In my past experience I worked on couple of tools like version control system GIT & Bitbucket.**

**Coming to Build tools I have very good experience in Maven**

**Coming to Continuous Integration/ Continuous Deployment I have very good experience in Jenkins and Basic knowledge on Bamboo.**

**I’m participating the couple of activities like installing the plugins and installing nodes and creating CI/CD Jenkins pipelines for my project requirements.**

**Configuration management tool I worked on Ansible and writing playbooks.**

**Proven experience in containerization tool docker and knowledge on kubernetes**

**Later coming to third party backup repository nexus I’m having very good experience**

**Coming to cloud services, I’m having experience in aws**

**Experience in creating multiple VPC’s and public and private subnets as per requirement and distribute them as groups into various availability zone of VPC**

**Experience in creating S3 buckets with various life cycle policies based on requirements**

**Experience in creating Linux agent and Integration of CI/CD for EKS cluster in AWS.**

**Experience on Infrastructure as a code using Cloud formation & terraform**

**Knowledge in creating Build/Release pipelines in Azure-DevOps**

**Knowledge in creation of Virtual machines, active directory and storage accounts**

**Experience in release managemet and atlassian tools like JIRA, bitbucket, confluence**

**Experience in SME , SDL process and creation on Agile boards and conducting scrum meetings.**

Evaluate the business processes and operations by implementing their subject expertise

 Experience in essentialities and requirements of software development and recommends some technical solutions.

Expertise in monitor the testing, development, and implementation of technical solutions.

**AZURE SELF INTRO:**

**My self manikanta, I’m having 5years of experience on devops and AWS**

**Past 5 years I have been working in carelon global organization as DevOps engineer**

**Well coming to my skillset:**

**In my past experience I worked on couple of tools like version control system GIT & Bitbucket.**

**Coming to Build tools I have very good experience in Maven**

**Coming to Continuous Integration/ Continuous Deployment I have very good experience in Jenkins and Basic knowledge on Bamboo.**

**I’m participating the couple of activities like installing the plugins and installing nodes and creating CI/CD Jenkins pipelines for my project requirements.**

**Configuration management tool I worked on Ansible and writing playbooks.**

**Proven experience in containerization tool Docker and knowledge on kubernetes**

**Coming to cloud services, I’m having experience in Azure and Knowledge on AWS**

**Experience in creating Build/Release pipelines in Azure-DevOps**

**Experience in creating Linux agent and Integration of CI/CD for AKS cluster in Azure.**

**Experience in setting up services, Active directory, Key vaults, Role based Access control(RBAC) in Azure.**

**Experience on Infrastructure as a code using Azure resource manager(ARM) and Terraform**

**Experience in Tokenize the data in snowflake and deploy Hadoop application in Cloud as Azure.**

**Experience in Infra/Unix Migrations and scheduling jobs in Control-M for ETLmetadata.**

**Knowledge in creating Storage accounts with various life cycle policies based on requirements**

**Experience in release managemet and atlassian tools like JIRA, bitbucket, confluence**

**Experience in SME , SDL process and creation on Agile boards and conducting scrum meetings.**

**DevOps Onpremisis( Monolithic project):**

**Developers develop the code in the format of java, and this source code is managed by version control system tool like GIT**

**Developers send this code to the Github repository and any changes made in the code is committed to this repository.**

**After Jenkins pull this code from the repository using the git plugin and automate using gitwebhooks and we are providing Jenkins URL in gitwebhook configuration and build it using tools like maven**

**After build triggers and build successful, Jenkins releases this code on the test environment (we have dev, UAT, staging and PROD environments)**

**Once code is tested , we have provided production server details in Jenkins.**

**Jenkins sent for deployment in production server using service iD**

**After deployment and it’s continuously monitoring by tools like splunk.**

**Azure-Devops project:**

**What is your day to day activities (or) roles and responsibilities in current project**

**I’m working on C bank project, Here we are following agile methodology and 2 weeks sprint, every 2 weeks we have production deployment, till 2 weeks my developers they are providing the code for deploying into dev environment, UAT and prod environments.**

**Once they provided the required branch details from the github repository then we will start for building the artifacts and deploying the artifacts to dev and UAT, once deployment is completed will go for live.**

**We used to create the request or tickets in JIRA canban boards, every day we have a standup call, in that we have to discuss what task we are doing currently and what task next I’m going to implement and what tasks is completed.**

**We have to update the complete status to the client.**

**This is the work culture we are following in our organization.**

**Jenkins or Azure-devops CI/CD pipeline:**

**My project is EDW project, My role us to take care of devops & Cloud activities, when it comes to regular activities closely working with developers.**

**Once they are developing the code then I need to make CI/CD pipeline.**

**For XXXX project, my application hosted in azure, we are using azure cloud where my application hosted in kubernetes**

**In current project we are maintaining azure AKS service for managed Kubernetes**

**Here how exactly we have integrated is through the Jenkins CI/CD pipeline we have implemented k8s service**

**Under the Jenkins we are writing the Jenkins file, the Jenkins file we are writing groovy code in declarative pipeline, With help of groovy we are specifying number of stages.**

**What are the stages are available here is Cloning the repository, Building the artifacts,**

**Checking the code quality and uploading to the artifact either nexus (or) docker hub.**

**Once the registry uploaded with help of AKS then we are deploying the connectivity b/w Azure(AKS) and Jenkins.**

**How we are giving connectivity is , Once we are created K8s cluster from azure we are downloading the .kube configuration file of my cluster and uploading to the repository at the time of cloning the complete repository,**

**I have defined the number of stages based on it will execute each stage and it will deploy into aks cluster.**

**AWS project :**

**Developers develop there code in the format of java, they will make a change on a feature branch in there local git repository**

**Push the change to your AWS code commit hosted repository and have the change trigger a merge from the feature branch to develop branch.**

**After perform a build on the merged develop branch using Jenkins after successful build for this we will run the Jenkins server on amazon liunx instance and configured the workstation to access the git repository hosted by aws code commit**

**We will setup IAM permissions to control access to the GIT Repositories.**

**We will associate the IAM role with the amazon Ec2 instance to launch to run Jenkins**

**After will configure AWS CLI to configure AWS in Jenkins**

**Cd Jenkins**

**Sudo –u Jenkins aws configure**

**Migration project:( terraform)**

**Actually it’s a migration project, we are getting TSD from the application team like technical specification document, so based on TSD we are going to write the terraform scripts in our local machine and then we will test the scripts.**

**Before that we are going to create feature branch in github repository and then we are pushing our script from local to github repository**

**So once we pushed to github repository, we are going to create CI/CD pipeline for executing terraform scripts,**

**Because we need to setup the infrastructure for multiple environments right, so based on requirement we are going to run the pipeline and we will setup the infrastructure.**

**Production support issues:**

[**https://applicationproductionsupport.wordpress.com/common-issues-and-solutions/**](https://applicationproductionsupport.wordpress.com/common-issues-and-solutions/)

**JOB change:**

* *I want to build on one of the aspects that I like most about the work I currently am doing…*
* *One of the things that has made things a little more challenging is that I’d like to have a platform where I could share my ideas and offer up ways to improve…(service, operations, technology, communication, etc…)*

## Continuous Integration (CI)

[Continuous Integration](https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment) is the practice where developers merge the changes to the code base to the main branch as often as possible. These changes are validated by creating a build and then running automated tests against the build. If these tests don’t pass, the changes aren’t merged,

### Benefits of Continuous Integration

This process also causes fewer bugs to be shipped to production as the issues are caught early and integration issues are solved before release.

## Continuous Delivery (CD)

[Continuous Delivery](https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment) is an extension of CI since it enables automation to deploy all the code changes to an environment (dev, qa, stage, prod, etc) after the changes have been merged. The artifact may be built as part of CI or as part of this process since the source of truth (your repository) is reliable given your CI proces

## Continuous Deployment (CD)

Continuous Deployment takes the process one step further than continuous delivery. Here, all changes that pass the verification steps at each stage in the pipeline are released to production. This process is completely automated and only a failed verification step will prevent pushing the changes to production.

### Benefits of Continuous Deployment

Apart from the fact that customers get updates quicker, developers also get feedback faster which means there is less pressure as small changes are pushed incrementally compared to big updates not that often

## Continuous Integration vs Continuous Deployment

The differences between continuous integration and continuous deployment are largely similar to the difference mentioned above. The further difference is that continuous deployment deploys changes to the customers automatically without any human intervention.

## Continuous Delivery vs Continuous Deployment

By this time, the difference here is obvious. Continuous delivery is a partly manual process where developers can deploy any changes to customers by simply clicking a button, while continuous deployment emphasizes automating the entire the process.

Is DevOps an Agile approach?

KEY DIFFERENCE. **DevOps is a practice of bringing development and operations teams together whereas Agile is an iterative approach that focuses on collaboration, customer feedback and small rapid releases**

**Agile focuses on optimizing the development life-cycle, while DevOps unites development and operations in a CI/CD environment**.

**What is load balancing?**

• Load Balances are servers that forward traffic to multiple

servers (e.g., EC2 instances) downstream.

**Why use a load balancer?**

• Spread load across multiple downstream instances

• Expose a single point of access (DNS) to your application

• Seamlessly handle failures of downstream instances

• Do regular health checks to your instances

• Provide SSL termination (HTTPS) for your websites

• Enforce stickiness with cookies

• High availability across zones

• Separate public traffic from private traffic

**Types of load balancer on AWS**

• AWS has 4 kinds of managed Load Balancers

• Classic Load Balancer (v1 - old generation) – 2009 – CLB

• HTTP, HTTPS, TCP, SSL (secure TCP)

• Application Load Balancer (v2 - new generation) – 2016 – ALB

• HTTP, HTTPS, WebSocket

• Network Load Balancer (v2 - new generation) – 2017 – NLB

• TCP, TLS (secure TCP), UDP

• Gateway Load Balancer – 2020 – GWLB

• Operates at layer 3 (Network layer) – IP Protocol

• Overall, it is recommended to use the newer generation load balancers as they

provide more features

• Some load balancers can be setup as internal (private) or external (public) ELBs

**High Availability & Scalability For EC2**

• **Vertical Scaling**: Increase instance size (= scale up / down)

• From: t2.nano - 0.5G of RAM, 1 vCPU

• To: u-12tb1.metal – 12.3 TB of RAM, 448 vCPUs

• **Horizontal Scaling:** Increase number of instances (= scale out / in)

• Auto Scaling Group

• Load Balancer

• High Availability: Run instances for the same application across multi AZ

• Auto Scaling Group multi AZ

• Load Balancer multi AZ

|  |  |  |
| --- | --- | --- |
|  | **Horizontal Scaling (scaling out)** | **Vertical Scaling (scaling up)** |
| **Databases** | In a database world, horizontal scaling is usually based on the partitioning of data (each node only contains part of the data). | In vertical scaling, the data lives on a single node and scaling is done through multi-core, e.g. spreading the load between the CPU and RAM resources of the machine. |
| **Downtime** | In theory, adding more machines to the existing pool means you are not limited to the capacity of a single unit, making it possible to scale with less downtime. | Vertical scaling is limited to the capacity of one machine, scaling beyond that capacity can involve downtime and has an upper hard limit, i.e. the scale of the hardware on which you are currently running. |

**What’s an Auto Scaling Group?**

• In real-life, the load on your websites and application can change

• In the cloud, you can create and get rid of servers very quickly

• The goal of an Auto Scaling Group (ASG) is to:

• Scale out (add EC2 instances) to match an increased load

• Scale in (remove EC2 instances) to match a decreased load

• Ensure we have a minimum and a maximum number of machines running

• Automatically Register new instances to a load balancer

Blue/Green Deployment

A blue/green deployment is a deployment strategy in which you create two separate, but identical environments. One environment (blue) is running the current application version and one environment (green) is running the new application version. Using a blue/green deployment strategy increases application availability and reduces deployment risk by simplifying the rollback process if a deployment fails. Once testing has been completed on the green environment, live application traffic is directed to the green environment and the blue environment is deprecated.

Blue-green deployment is a technique that reduces downtime and risk by running two identical production environments called Blue and Green.

At any time, only one of the environments is live, with the live environment serving all production traffic. For this example, Blue is currently live and Green is idle.

As you prepare a new version of your software, deployment and the final stage of testing takes place in the environment that is not live: in this example, Green. Once you have deployed and fully tested the software in Green, you switch the router so all incoming requests now go to Green instead of Blue. Green is now live, and Blue is idle.

Link: <https://docs.cloudfoundry.org/devguide/deploy-apps/blue-green.html>

how to change the public & private ip address in aws

<https://aws.amazon.com/premiumsupport/knowledge-center/static-private-ip-windows/>

how to change security group in aws

<https://docs.aws.amazon.com/vpc/latest/userguide/VPC_SecurityGroups.html>

Maven life cycle goals:

the default lifecycle comprises of the following phases (for a complete list of the lifecycle phases, refer to the [Lifecycle Reference](https://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html#Lifecycle_Reference)):

* validate - validate the project is correct and all necessary information is available
* compile - compile the source code of the project
* test - test the compiled source code using a suitable unit testing framework. These tests should not require the code be packaged or deployed
* package - take the compiled code and package it in its distributable format, such as a JAR.
* verify - run any checks on results of integration tests to ensure quality criteria are met
* install - install the package into the local repository, for use as a dependency in other projects locally
* deploy - done in the build environment, copies the final package to the remote repository for sharing with other developers and projects.

GIT:

Diff b/w git pull & stash & fetch

### Fetch

$ git fetch origin

**git fetch** really only downloads new data from a remote repository - but it doesn't integrate any of this new data into your working files. Fetch is great for getting a fresh view on all the things that happened in a remote repository.  
Due to it's "harmless" nature, you can rest assured: fetch will never manipulate, destroy, or screw up anything. This means you can never fetch often enough.

### Pull

$ git pull origin master

**git pull**, in contrast, is used with a different goal in mind: to update your current HEAD branch with the latest changes from the remote server. This means that pull not only downloads new data; it also directly **integrates** it into your current working copy files. This has a couple of consequences:

* Since "git pull" tries to merge remote changes with your local ones, a so-called "merge conflict" can occur. Check out our in-depth tutorial on [How to deal with merge conflicts](https://www.git-tower.com/learn/git/ebook/en/command-line/advanced-topics/merge-conflicts) for more information.
* Like for many other actions, it's highly recommended to start a "git pull" only with a clean working copy. This means that you should not have any uncommitted local changes before you pull. Use Git's Stash feature to [save your local changes temporarily](https://www.git-tower.com/learn/git/ebook/en/command-line/branching-merging/stashing).

GIT Stash:

Git stash is a built-in command with the distributed Version control tool in [Git](https://www.techtarget.com/searchitoperations/definition/Git) that locally stores all the most recent changes in a workspace and resets the state of the workspace to the prior commit state.

A user can retrieve all files put into the stash with the git stash pop and git stash apply commands. Git stash acts as a mechanism to locally version files without those versions being seen by other developers who share the same git repository

Cron tab:

<https://www.adminschoice.com/crontab-quick-reference>

# ┌───────────── minute (0 - 59)

# │ ┌───────────── hour (0 - 23)

# │ │ ┌───────────── day of the month (1 - 31)

# │ │ │ ┌───────────── month (1 - 12)

# │ │ │ │ ┌───────────── day of the week (0 - 6) (Sunday to Saturday;

# │ │ │ │ │ 7 is also Sunday on some systems)

# │ │ │ │ │

# │ │ │ │ │

# \* \* \* \* \* <command to execute>

**Docker :**

**Monolithic application:** Under the monolithic application when you want to restart your application that application hosted in host server, when you want to perform any activity, the entire Base OS should be restart.

**Micro service application:** No need to reboot the complete application, that particular container you can stop, no need to stop complete base os, each OS I will consider as container**.**

**Micro service application architecture:**

[**https://middleware.io/blog/microservices-architecture/**](https://middleware.io/blog/microservices-architecture/)

Docker commands:

<URL:edureka.co/blog/docker-commands/>

<https://towardsdatascience.com/12-essential-docker-commands-you-should-know-c2d5a7751bb5>

## Dockerfile Basics

Before we construct our Dockerfile, you need to understand what makes up the file. This will be a text file, named *Dockerfile*, that includes specific keywords that dictate how to build a specific image. The specific keywords you can use in a file are:

* **ADD** copies the files from a source on the host into the container’s own filesystem at the set destination.
* **CMD** can be used for executing a specific command within the container.
* **ENTRYPOINT** sets a default application to be used every time a container is created with the image.
* **ENV** sets environment variables.
* **EXPOSE** associates a specific port to enable networking between the container and the outside world.
* **FROM** defines the base image used to start the build process.
* **MAINTAINER** defines a full name and email address of the image creator.
* **RUN** is the central executing directive for Dockerfiles.
* **USER** sets the UID (or username) which is to run the container.
* **VOLUME** is used to enable access from the container to a directory on the host machine.
* **WORKDIR** sets the path where the command, defined with CMD, is to be executed.
* **LABEL**allows you to add a label to your docker image.

Not all keywords are required for a Dockerfile to function. Case in point, our example will only make use of **FROM**, **MAINTAINER**, and **RUN**.

Docker file syntax:

# syntax=docker/dockerfile:1

FROM node:12-alpine

RUN apk add --no-cache python2 g++ make

WORKDIR /app

COPY . .

RUN yarn install --production

CMD ["node", "src/index.js"]

EXPOSE 3000

## Step 2: Create a Dockerfile

In this step, you write a Dockerfile that builds a Docker image. The image contains all the dependencies the Python application requires, including Python itself.

In your project directory, create a file named Dockerfile and paste the following:

|  |
| --- |
| FROMcentos:7 |
|  | MAINTAINER The CentOS Project <cloud-ops@centos.org> |
|  |  |
|  | RUN yum -y install docker-distribution; yum clean all |
|  |  |
|  | EXPOSE 5000 |
|  |  |
|  | USER 42 |
|  |  |
|  | ENTRYPOINT ["/usr/bin/registry"] |
|  | CMD ["/etc/docker-distribution/registry/config.yml"] |

FROM ubuntu:18.04

RUN apt-get update && \

apt-get install -y redis-server && \

apt-get clean

EXPOSE 6379

CMD ["redis-server", "--protected-mode no"]

# syntax=docker/dockerfile:1

FROM python:3.7-alpine

WORKDIR /code

ENV FLASK\_APP=app.py

ENV FLASK\_RUN\_HOST=0.0.0.0

RUN apk add --no-cache gcc musl-dev linux-headers

COPY requirements.txt requirements.txt

RUN pip install -r requirements.txt

EXPOSE 5000

COPY . .

CMD ["flask", "run"]

This tells Docker to:

* Build an image starting with the Python 3.7 image.
* Set the working directory to /code.
* Set environment variables used by the flask command.
* Install gcc and other dependencies
* Copy requirements.txt and install the Python dependencies.
* Add metadata to the image to describe that the container is listening on port 5000
* Copy the current directory . in the project to the workdir . in the image.
* Set the default command for the container to flask run.

**Kubernets deployment file:**

apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx-deployment

spec:

replicas: 1

selector:

matchLabels:

app: nginx-app

template:

metadata:

labels:

app: nginx-app

spec:

containers:

- name: nginx-container

image: nginx

---

apiVersion: v1

kind: Service

metadata:

name: nginx-service

spec:

ports:

- name: http

targetPort: 80

port: 80

selector:

app: nginx-app

type: LoadBalancer

**Diff b/w Docker Swarm & Kubernetes**

Docker Swarm is a lightweight, easy-to-use orchestration tool with limited offerings compared to Kubernetes. In contrast, Kubernetes is complex but powerful and provides self-healing, auto-scaling capabilities out of the box

**Docker networking issues and troubleshooting**

[**https://birthday.play-with-docker.com/troubleshooting-network-issues/**](https://birthday.play-with-docker.com/troubleshooting-network-issues/)

**Docker file issues and troubleshoot:**

[**https://docs.docker.com/desktop/windows/troubleshoot/**](https://docs.docker.com/desktop/windows/troubleshoot/)

**how to backup of your container if it is failes**

## backup docker images

To backup docker images, use the [docker save](https://docs.docker.com/engine/reference/commandline/save) command that will produce a tar archive that can be used later on to create a new docker image with the [docker load](https://docs.docker.com/engine/reference/commandline/load/) command.

## backup docker containers

You can backup a docker container by different means

* by committing a new docker image based on the docker container current state using the [docker commit](https://docs.docker.com/engine/reference/commandline/commit/) command
* by exporting the docker container file system as a tar archive using the [docker export](https://docs.docker.com/engine/reference/commandline/export/) command. You can later on create a new docker image from that tar archive with the [docker import](https://docs.docker.com/engine/reference/commandline/import/) command.

Be aware that those commands will only backup the docker container layered file system. **This excludes the data volumes**.

## backup docker data volumes

To backup a data volume you can run a new container using the volume you want to backup and executing the tar command to produce an archive of the volume content as described in the [docker user guide](https://docs.docker.com/storage/volumes/" \l "backup-restore-or-migrate-data-volumes).

In your particular case, the data volume is used to store the data for a MySQL server. So if you want to export a tar archive for this volume, you will need to stop the MySQL server first. To do so you will have to stop the wordpress container.

**Docker Volumes:**

Volumes are the preferred mechanism for persisting data generated by and used by Docker containers. While [bind mounts](https://docs.docker.com/storage/bind-mounts/) are dependent on the directory structure and OS of the host machine, volumes are completely managed by Docker. Volumes have several advantages over bind mounts:

* Volumes are easier to back up or migrate than bind mounts.
* You can manage volumes using Docker CLI commands or the Docker API.
* Volumes work on both Linux and Windows containers.
* Volumes can be more safely shared among multiple containers.
* Volume drivers let you store volumes on remote hosts or cloud providers, to encrypt the contents of volumes, or to add other functionality.
* Create a volume
* $ docker volume create [OPTIONS] [VOLUME]
* docker volume create hello
* hello
* $ docker run -d -v hello:/world busybox ls /world

# docker manifest:

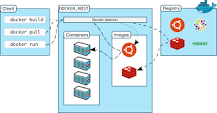
The docker manifest command by itself performs no action. In order to operate on a manifest or manifest list, one of the subcommands must be used.

A single manifest is information about an image, such as layers, size, and digest. The docker manifest command also gives users additional information such as the os and architecture an image was built for.

A manifest list is a list of image layers that is created by specifying one or more (ideally more than one) image names. It can then be used in the same way as an image name in docker pull and docker run commands, for example.

### manifest inspect , manifest create

**What is a Docker namespace?**

[[](https://www.google.com/search?rlz=1C1GCEJ_enIN974IN974&sxsrf=ALiCzsapA-QcSeGJDMfF93C-tqV18oQaUQ:1655868500139&q=What+is+a+Docker+namespace?&tbm=isch&source=iu&ictx=1&vet=1&fir=ozZsDDX9ZCaf4M,3Rlh9Be8h2ItMM,_&usg=AI4_-kR4jS6uiaP5Zll7eGDpZkrO6PDLxw&sa=X&ved=2ahUKEwjA9frtjsD4AhVf1jgGHcSdBKMQ9QF6BAgcEAE#imgrc=ozZsDDX9ZCaf4M)](https://www.google.com/search?rlz=1C1GCEJ_enIN974IN974&sxsrf=ALiCzsapA-QcSeGJDMfF93C-tqV18oQaUQ:1655868500139&q=What+is+a+Docker+namespace?&tbm=isch&source=iu&ictx=1&vet=1&fir=ozZsDDX9ZCaf4M%252C3Rlh9Be8h2ItMM%252C_&usg=AI4_-kR4jS6uiaP5Zll7eGDpZkrO6PDLxw&sa=X&ved=2ahUKEwjA9frtjsD4AhVf1jgGHcSdBKMQ9QF6BAgcEAE" \l "imgrc=ozZsDDX9ZCaf4M)

Docker uses a technology called namespaces **to provide the isolated workspace called the container**. When you run a container, Docker creates a set of namespaces for that container. These namespaces provide a layer of isolation.

**Namespace Types:**

1. Process ID
2. Mount
3. IPC (Interprocess communication)
4. User (currently experimental support for)
5. Network

Kubernetes:

1.What is Kubernetes autoscaling?

"Kubernetes autoscaling **helps optimize resource usage and costs by automatically scaling a cluster up and down in line with demand**." “Kubernetes autoscaling helps optimize resource usage and costs by automatically scaling a cluster up and down in line with demand

<https://kubernetes.io/blog/2016/07/autoscaling-in-kubernetes/>

2. Components to create a pod

## A pod is a collection of containers and its storage inside a node of a Kubernetes cluster. It is possible to create a pod with multiple containers inside it.

## Types of Pod

There are two types of Pods −

* Single container pod
* Multi container pod

### Single Container Pod

They can be simply created with the kubctl run command, where you have a defined image on the Docker registry which we will pull while creating a pod.

$ kubectl run <name of pod> --image=<name of the image from registry>

**Example** − We will create a pod with a tomcat image which is available on the Docker hub.

$ kubectl run tomcat --image = tomcat:8.0

This can also be done by creating the **yaml** file and then running the **kubectl create** command.

apiVersion: v1

kind: Pod

metadata:

name: Tomcat

spec:

containers:

- name: Tomcat

image: tomcat: 8.0

ports:

containerPort: 7500

imagePullPolicy: Always

**What is Kubernetes ReplicaSet?**

A ReplicaSet (RS) is a Kubernetes object that ensures there is always a stable set of running pods for a specific workload. The ReplicaSet configuration defines a number of identical pods required, and if a pod is evicted or fails, creates more pods to compensate for the loss.

**How is Kubernetes used for scalability?**  
  
**Kubernetes allows users to horizontally scale the total containers used based on the application requirements**, which may change over time. It's easy to change the number via the command line

**What is key-value in Kubernetes?**

**Kubernetes uses etcd as a key-value database store**. It stores the configuration of the Kubernetes cluster in etcd. It also stores the actual state of the system and the desired state of the system in etcd

**Kubernetes control manager:**

The Kubernetes controller manager is **a daemon that embeds the core control loops shipped with Kubernetes**. In applications of robotics and automation, a control loop is a non-terminating loop that regulates the state of the system

## kube controller manager : Some types of controllers

### ****1.Node Controller****

### ****2. Replica Controller****

### ****3. Job Controller****

### ****4. Deployment Controller****

<https://blog.knoldus.com/introduction-to-kube-controller-manager/>

What is minikube?

minikube. Like kind , minikube is a tool that **lets you run Kubernetes locally**. minikube runs a single-node Kubernetes cluster on your personal computer (including Windows, macOS and Linux PCs) so that you can try out Kubernetes, or for daily development work.

**Terraform modules:**

1. Root Module:

The root module is **the directory that holds the Terraform configuration files that are applied to build your desired infrastructure**. These files provide an entry point into any nested modules you might utilize. Any module should include, at minimum, a main.tf , a variables.tf , and an outputs.tf file.

1. Child Module

A Terraform module (usually the root module of a configuration) can call other modules to include their resources into the configuration. **A module that has been called by another module** is often referred to as a child module.

Creating modules:

<https://www.terraform.io/language/modules/develop>

Terraform commands:

Terraform Providing providers to build infrastructure in any cloud platform like AWS, Azure, GCP

**Terraform state file**:

Terraform have feature to maintain the state file with help of state file we will reuse the resources which we have build already.

It contain details about infrastructure which we have build, we can store state file for backup and we can use the same for infracture creation

Terraform having 2 files:

Main.tf -- resources

Variable.tf – aws accesskeys

**Terraform init** – It will download the required plugins for your resource

**Terraform plan** – It will be check what are resources your going to be create (or) destroy. While running terraform plan, it will create automatically terraform state file.

**Terraform apply** – It will create the resources.

**Terraform destroy** – It will be deleting the resources from your account.

**LINUX:**

**Public DNS/IP:** It is allocated from a pool of available IP's and it is mandatory to let you connect from anywhere around the globe to your EC2 instance.

**Private IP:** Its allocation is based on vpc/subnet in which EC2 is setup. Every subnet has a range of IP's, out of which one IP gets allocated to the launched EC2. Scope or visibility of this IP is only under the defined VPC. Hence, to communicate between two or more EC2 instances using private IP, all must be under the same vpc.

**Note:** Private IP designated to an EC2 remains same until vpc is same.

**Elastic IP:** - It is similar to static IP and can be assign to any EC2 instance. Once we assign it, existing public IP gets released and replaced with the newly assigned Elastic IP. They are allocated to the AWS account so that we can release it from specific EC2 and re-assign it to any other EC2 instances (if needed).

Diff b/w public and private subnets:

**The instances in the public subnet can send outbound traffic directly to the internet, whereas the instances in the private subnet can't**. Instead, the instances in the private subnet can access the internet by using a network address translation (NAT) gateway that resides in the public subnet.

**A public subnet is a subnet that is associated with a route table that has a route to an Internet gateway**. This connects the VPC to the Internet and to other AWS services. Private Subnet. A private subnet is a subnet that is associated with a route table that doesn't have a route to an internet gateway.

**How to create user in linux:**

1. Launch a terminal application.
2. Run adduser command with a username as argument.

$ sudo adduser username

1. Enter password for current user if necessary.

[sudo] password for user:

**What is use case of kubernetes name space:**