Devops-AWS-Docker-K8s

- Raghavaiah Avula

Note: This training class is for community service(non profit orgs) for knowledge sharing

Myself …

Name: Raghavaiah Avula

LinkedIn: https://www.linkedin.com/in/raghavaiah
GitHub: https://github.com/raavula

Email: reachraghav10@gmail.com

Note: For this course refer below repo for practice example
https://github.com/raavula/devops

Devops What we can discuss ?

- AWS Service
- Monitoring
- Logging
- CI/CD
- Deployment
- Scale and performance
- Security
- Databases
- Tools
- Scripts and Automation

Lets start with Network services …

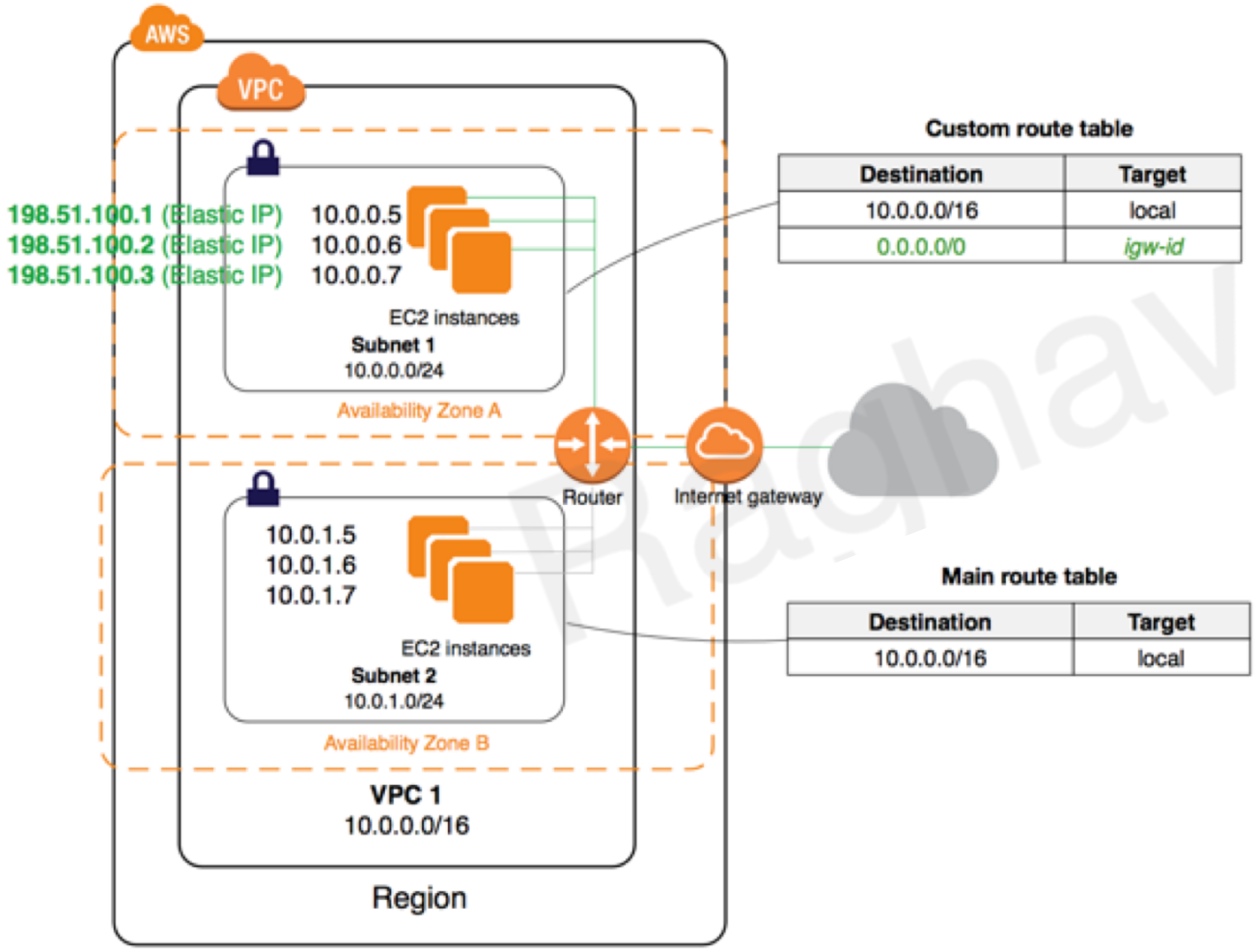
AWS Networking Products

Service	Product Type	Description
Amazon CloudFront	Content Delivery Network (CDN)	Highly secure global CDN to get content to your viewers with low latency and high transfer speeds
Amazon VPC	Virtual Private Cloud	Isolate cloud resources with your own private virtual network
AWS Direct Connect	Dedicated Network Connections to AWS	Dedicated network connection between your network and your Amazon VPC
Elastic Load Balancing	Load Balancing	Automatically distribute application traffic across multiple Amazon EC2 instances in the cloud
Amazon Route 53	Domain Name Service (DNS)	Highly available and scalable cloud DNS to connect user requests to your AWS resources

Others …

- PrivateLink
- API Gateway

AWS – VPC Service



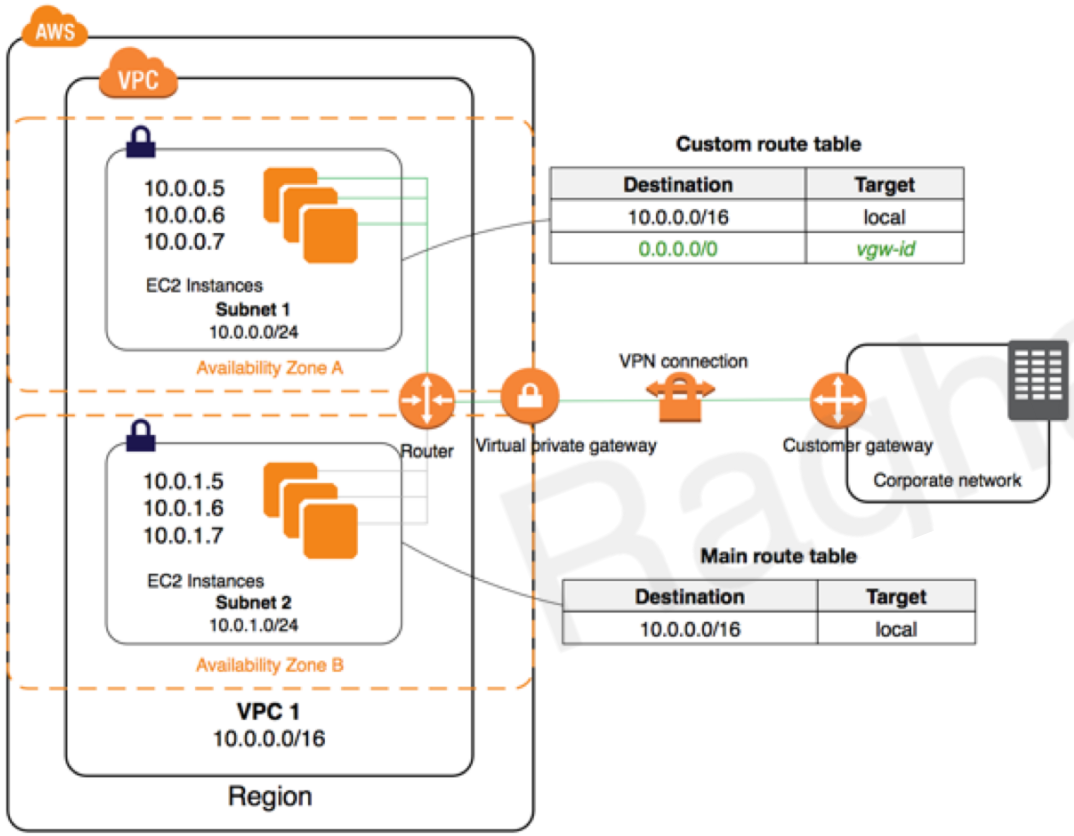
Types:

- vpc with single public subnet
- vpc with public and private subnet(NAT)
- vpc with pub and priv subnet and aws managed vpn access
- vpc with private subnet only and aws managed vpn access

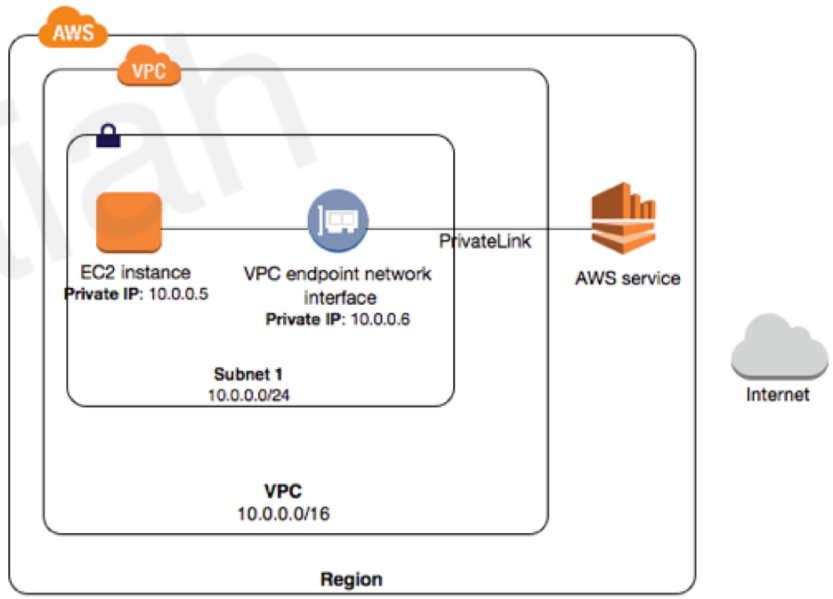
Notes:

- supports both ipv4 (CIDR: a/16) and ipv6 (CIDR: a/56)
- Migration to ipv6 support but ipv4 can't disable and act ad dual stack
- SG vs ACL
- Flow logs to cloud watch and s3
- RT ,IG , Egress-only IG(IPv6) vs NAT (ipv4)
- Provided DHCP and DNS
- Vpc peering
- Vpc endpoints powered by PrivateLink [interface = elastic network(cw,sns,apigw ..etc) & gateway = route in RT (s3,dyanamo db)]
- ClassicLink allows you to link an EC2-Classic instance to a VPC
- Vpn – aws managed vpn(IPsec), cloudHub,thirdpary vpn
- Last but not least – Limits apply ☺

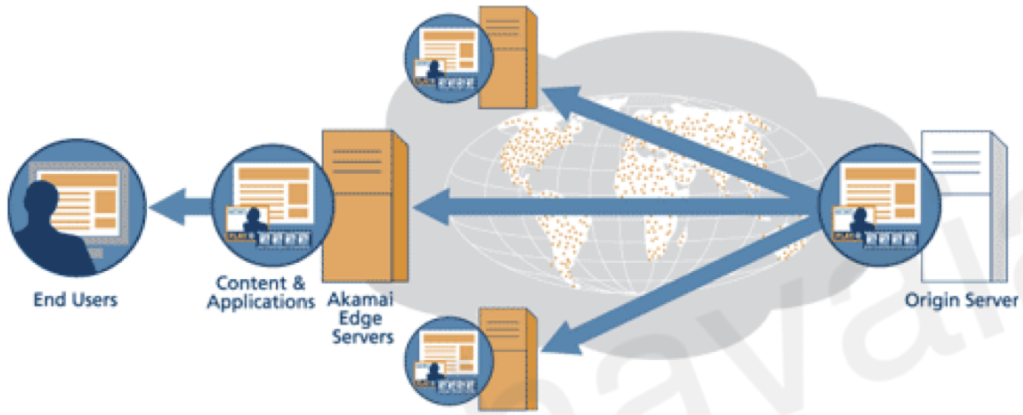
VPN case



Vpc Endpoint case



AWS – CloudFront service



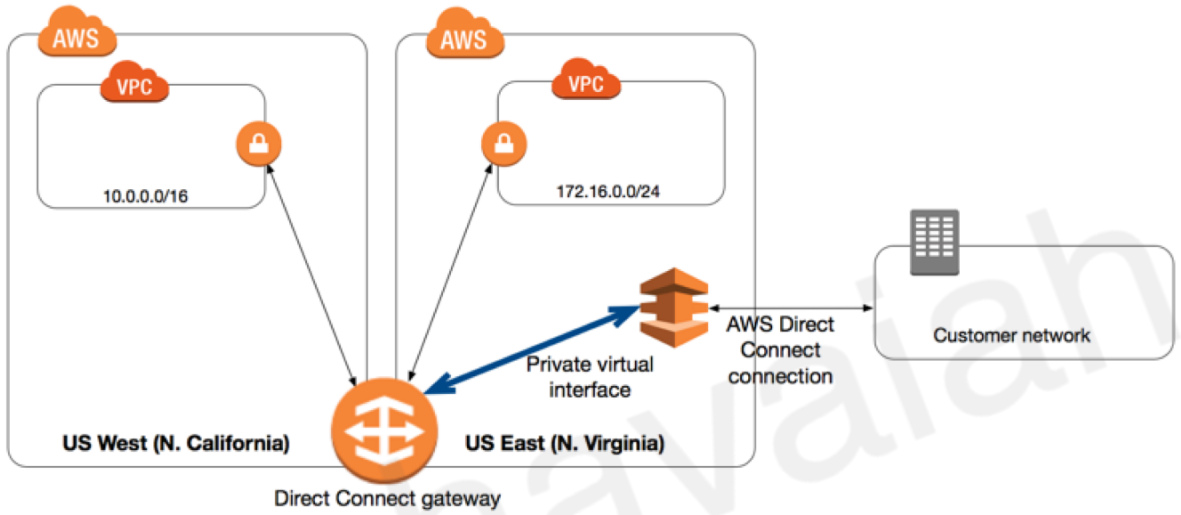
Benefits

- Global, Growing Content Delivery Network
- Secure Content at the Edge
- Programmable CDN (Content Deliver Network)
- High Performance
- Cost Effective
- Deep Integration with Key AWS Services

Use cases

- Static Asset Caching
- Live and On-Demand Video Streaming
- Security and DDoS Protection
- Dynamic and Customized Content
- API Acceleration
- Software Distribution

AWS – Direct Connect service



It easy to establish a dedicated network connection from your premises to AWS

Using industry standard 802.1q VLANs, this dedicated connection can be partitioned into multiple virtual interfaces

Benefits

- REDUCES YOUR BANDWIDTH COSTS
- CONSISTENT NETWORK PERFORMANCE
- COMPATIBLE WITH ALL AWS SERVICES
- PRIVATE CONNECTIVITY TO YOUR AMAZON VPC
- ELASTIC
- SIMPLE
- LARGE DATA SETS
- REAL TIME DATA FEED
- HYBRID ENVIRONMENT

AWS - Route 53 service

- Highly available and scalable cloud Domain Name System (DNS) web service.
- Amazon Route 53 effectively connects user requests to infrastructure running in AWS – such as Amazon EC2 instances, Elastic Load Balancing load balancers, or Amazon S3 buckets – and can also be used to route users to infrastructure outside of AWS.
- configure DNS health checks to route traffic to healthy endpoints or to independently monitor the health of your application and its endpoints
- Traffic Flow makes it easy for you to manage traffic globally through a variety of routing types, including Latency Based Routing, Geo DNS, Geoproximity, and Weighted Round Robin—all of which can be combined with DNS Failover in order to enable a variety of low-latency, fault-tolerant architectures

AWS – API Gateway service

- Easy for developers to publish, maintain, monitor, and secure APIs at any scale
- Build, deploy, and manage APIs
- Resiliency
- API Lifecycle Management
- SDK Generation
- API Operations Monitoring
- AWS Authorization
- API Keys for Third-Party Developers

AWS Compute service

Amazon EC2
Virtual servers in the cloud

Amazon EC2 Auto Scaling
Scale compute capacity to meet demand

Amazon EC2 Container Registry
Store and retrieve docker images

Amazon Elastic Container Service
Run and manage docker containers

Amazon Elastic Container Service for Kubernetes
Run managed Kubernetes on AWS

Amazon Lightsail
Launch and manage virtual private servers

AWS Batch
Run batch jobs at any scale

AWS Elastic Beanstalk
Run and manage web apps

AWS Fargate
Run containers without managing servers or clusters

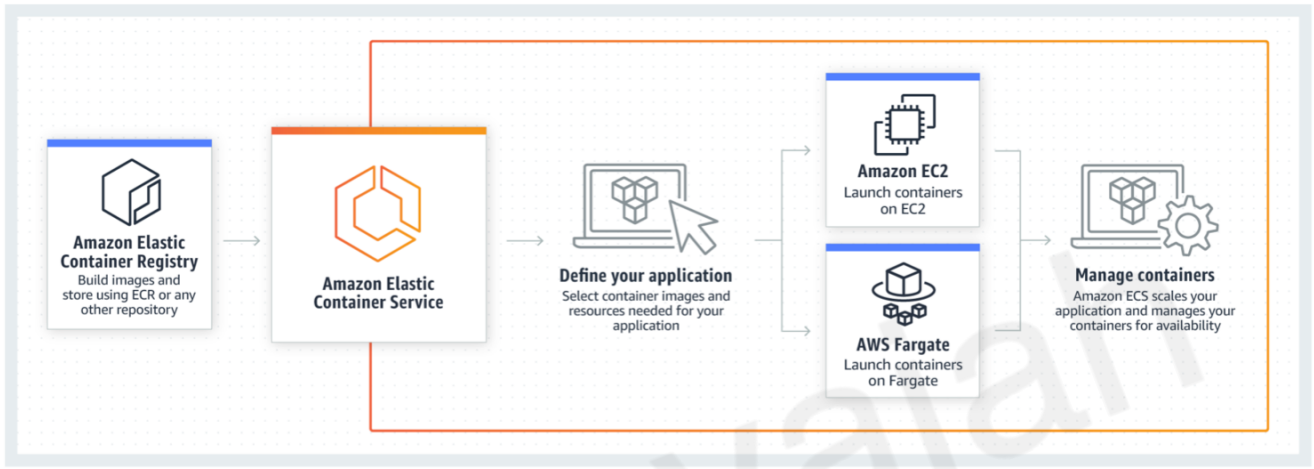
AWS Lambda
Run code without thinking about servers

AWS Serverless Application Repository
Discover, deploy, and publish serverless applications

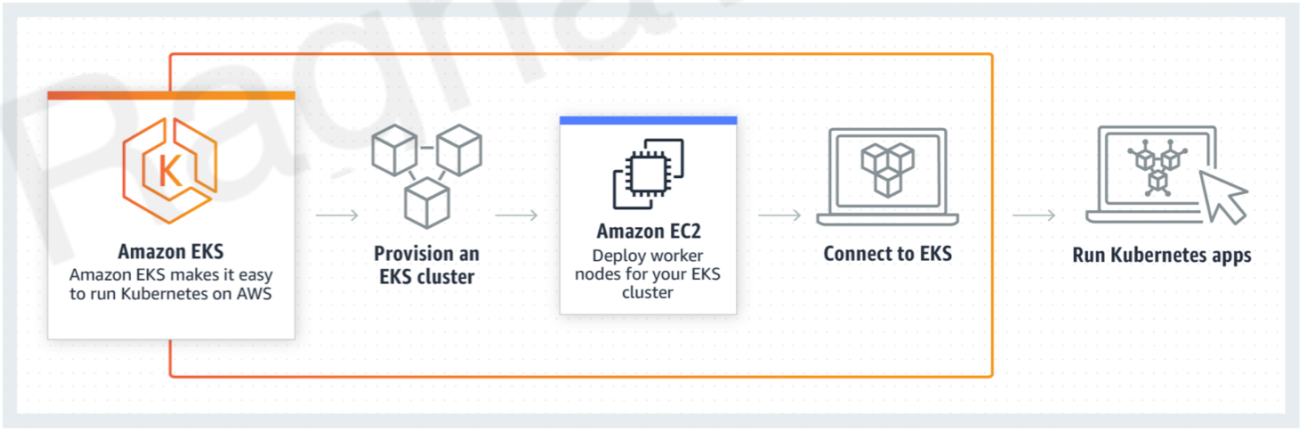
Elastic Load Balancing (ELB)
Distribute incoming traffic across multiple targets

VMware Cloud on AWS
Build a hybrid cloud without custom hardware

How Amazon ECS works



How Amazon EKS works



Notes

- Lightsail
  - manage vm
  - manage apps
- Beanstalk
  - Web server Env(elb)
  - Worker Env (asg)
- Batch
  - submit job queue
  - run jobs
  - monitor
- ECS
  - create cluster
  - task definition
  - ec2/Fargate
- Lambda
  - create function
  - server less Repo

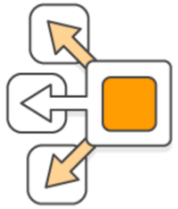
Elastic
Load Balancer

AWS – Elastic Load Balancing service

Application Load Balancer

- Layer-7 Load Balancing
- HTTPS Support
- Server Name Indication (SNI)
- IP addresses as Targets
- Content-Based Routing (Host & Path)
- Containerized Application Support
- HTTP/2 ,web socket and native IPv6 Support
- Slow Start Mode with Load-Balancing Algorithm
- User Authentication
- Redirects
- Logging – Access Logs

Network Load Balancer



- Connection-based Load Balancing(TCP)
- DNS Fail-over
- Integration with AWS Services
- Long-lived TCP Connections
- Central API Support
- Zonal Isolation
- Load Balancing using IP addresses as Targets
- Health Checks
- Preserve source IP address

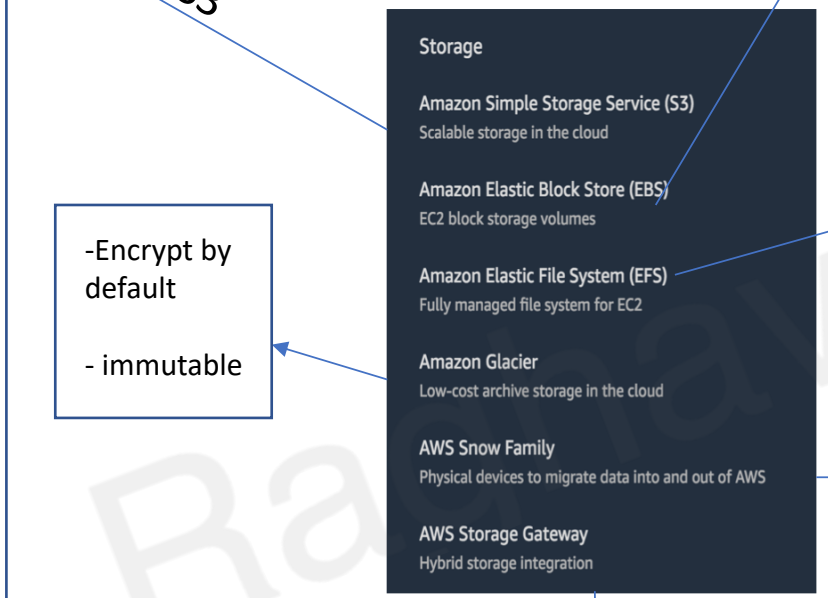
Classic Load Balancer

Layer 4 or Layer 7 Load Balancing
IPv6 Support
SSL Offloading
Sticky Sessions
Operational Monitoring
Logging
High Availability
Health Checks
Security Features

Elastic Load Balancer

Feature	Application Load Balancer	Network Load Balancer	Classic Load Balancer
Protocols	HTTP, HTTPS	TCP	TCP, SSL, HTTP, HTTPS
Platforms	VPC	VPC	EC2-Classic, VPC
Health checks	✔	✔	✔
CloudWatch metrics	✔	✔	✔
Logging	✔	✔	✔
Zonal fail-over	✔	✔	✔
Connection draining (deregistration delay)	✔	✔	✔
Load Balancing to multiple ports on the same instance	✔	✔	
WebSockets	✔	✔	
IP addresses as targets	✔	✔	
Load balancer deletion protection	✔	✔	
Path-Based Routing	✔		
Host-Based Routing	✔		
Native HTTP/2	✔		
Configurable idle connection timeout	✔		✔
Cross-zone load balancing	✔	✔	✔
SSL offloading	✔		✔
Server Name Indication (SNI)	✔		
Sticky sessions	✔		✔
Back-end server encryption	✔		✔
Static IP		✔	
Elastic IP address		✔	
Preserve Source IP address		✔	
Resource-based IAM permissions	✔	✔	✔
Tag-based IAM permissions	✔	✔	
Slow start	✔		
User authentication	✔		
Redirects	✔		
Fixed response	✔		

Storage service



S3

- classes: S3 Standard, S3 Standard-IA, S3 One Zone-IA, and Amazon Glacier

- security & Access Management
  - versioning
  - vpc endpoint
  - audit logs
  - mfa delete
  - time limit access to object

- Query in Place
  - s3 select (library like boto3)
  - Amazon Athena (server less)
  - Amazon Redshift spectrum

(open data format – Avro, cvs, test)

- Storage Management
  - tagging
  - versioning
  - cross region replication
  - life cycle management
  - class analysis
  - cloud watch and trail support

- Data Transfer
  - S3 Transfer Acceleration
  - Snowball-edge-mobile
  - storage gateway

EBS

- Volume type: **ssd**(iops, general) or **hdd**(throughput optimized, cold)
- API name: io1,gp2,st1,sc1
- Elastic volumes
- Ebs snapshots
- Ebs encryption and IAM access

EFS

- distributive across storage servers
- concurrently access data
- Elastic capacity
- Encryption

-Encrypt by default

- immutable

Snow

- moving data from on-prem to cloud
- 1 Gb n/w connection 100 PB takes 100 days
- snowball provides physical device with 3 types
  - snowball
  - snowball Edge
  - Snowmobile

- Hybrid cloud storage service that connects your existing on-premises environments with the AWS Cloud.
- Types: File Gateway, Volume Gateway, Tape Gateway
- Available in Pre install in a Hardware appliance

Database

-MySQL, PostgreSQL compatible RDB
- 5x MySQL, 3XPostgreSQL
- scale 64TB per instance
- backup to S3
- replication 3 AZ

Amazon Aurora
High performance managed relational database

Amazon DynamoDB
Managed NoSQL database

Amazon ElastiCache
In-memory caching service

Amazon Neptune
Fully managed graph database service

Amazon RDS
Managed relational database service for MySQL, PostgreSQL, Oracle, SQL Server, and MariaDB

Amazon Redshift
Fast, simple, cost-effective data warehousing

AWS Database Migration Service
Migrate databases with minimal downtime

- offers Redis and Memcached
- Redis works with Redis client for data store
- Memcached is in-memory-key-value store

- non relational database
- In memory cache
- key-value and document data model
- develop locally
- secondary index (column query)
- Streams is a time-ordered sequence of item-level changes
- triggers with aws lambda
- encryption at rest
- vpc endpoint

- fully managed graph database service
- supports popular graph models Property Graph and W3C's RDF, and their respective query languages Apache TinkerPop Gremlin and SPARQL
- example social-networking, fraud detection

- Relational database in cloud
- six familiar database engines including Arora

- extends data warehouse queries
- run analytics query on data (s3)
- massively parallel
- machine learning & caching
- Automated backup to s3
- end-to-end encryption
- network isolation(run in vpc)
- support cloud trail for audit logs

Analytics

Amazon Athena
Query data in S3 using SQL

Amazon CloudSearch
Managed search service

Amazon EMR
Hosted Hadoop framework

Amazon Elasticsearch Service
Run and scale Elasticsearch clusters

Amazon Kinesis
Analyze real-time video and data streams

Amazon Redshift
Fast, simple, cost-effective data warehousing

Amazon QuickSight
Fast business analytics service

AWS Data Pipeline
Orchestration service for periodic, data-driven workflows

AWS Glue
Prepare and load data

Security, Identity & compliance

AWS Identity and Access Management (IAM)
Manage user access and encryption keys

Amazon Cloud Directory
Create flexible cloud-native directories

Amazon Cognito
Identity management for your apps

Amazon GuardDuty
Managed threat detection service

Amazon Inspector
Analyze application security

Amazon Macie
Discover, classify, and protect your data

AWS Artifact
On-demand access to AWS' compliance reports

AWS Certificate Manager
Provision, manage, and deploy SSL/TLS certificates

AWS CloudHSM
Hardware-based key storage for regulatory compliance

AWS Directory Service
Host and manage active directory

AWS Firewall Manager
Central management of firewall rules

AWS Key Management Service
Managed creation and control of encryption keys

AWS Organizations
Policy-based management for multiple AWS accounts

AWS Secrets Manager
Rotate, manage, and retrieve secrets

AWS Shield
DDoS protection

AWS Single Sign-On
Cloud single sign-on (SSO) service

AWS WAF
Filter malicious web traffic

Management Tools

Amazon CloudWatch
Monitor resources and applications

AWS Auto Scaling
Scale multiple resources to meet demand

AWS CloudFormation
Create and manage resources with templates

AWS CloudTrail
Track user activity and API usage

AWS Command Line Interface
Unified tool to manage AWS services

AWS Config
Track resources inventory and changes

AWS Management Console
Web-based user interface

AWS Managed Services
Infrastructure operations management for AWS

AWS OpsWorks
Automate operations with Chef and Puppet

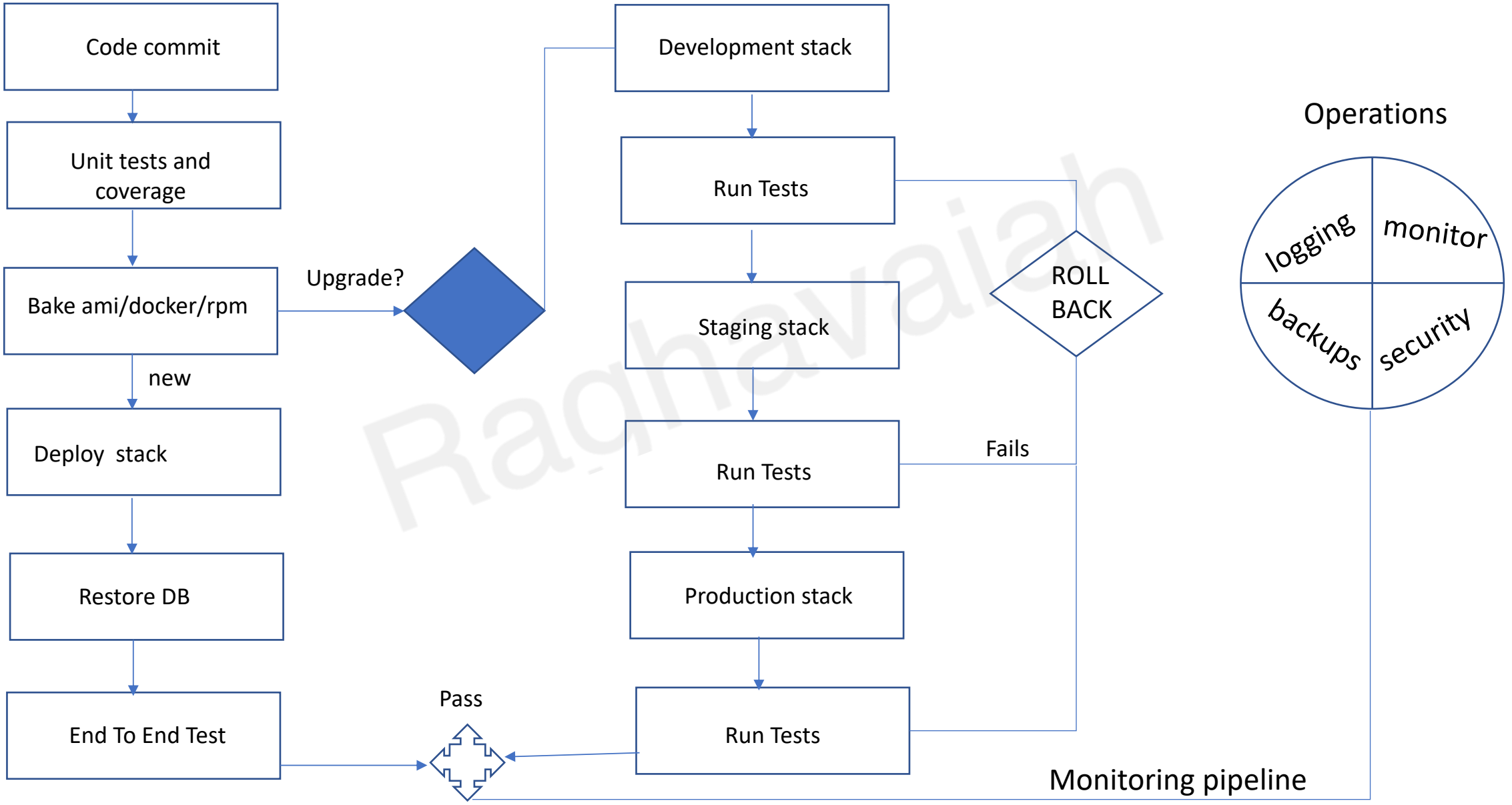
AWS Personal Health Dashboard
Personalized view of AWS service health

AWS Service Catalog
Create and use standardized products

AWS Systems Manager
Gain operational insights and take action

AWS Trusted Advisor
Optimize performance and security

CI/CD - Tools



Tools List

CI/CD phase	Tools
code commit / review	GitHub, bit bucket
Pipeline	Jenkins, aws CodePipleline, Spinnaker
Unit tests and coverage	Took based on application e.g. junit, code cover etc.
Bake ami/docker image/Rpms	Packer, Jfrog artifactory
Stack deployment (in aws, azure etc.)	Terraform
Stack configuration	Ansible,Puppet,Chef,SaltStack,consul,collect.d etc.
Certs and passwords	Vault
security	Twist lock, Sysdig security, app check, Qualys scanner, snort, aws security services(IAM etc.)
Deployment testing	The-Netflix-simian-army Tools (Monkey Tools)
Monitoring	Datadog,Nagios,Promethues,Grafana, AWS cloud watch metrics/alarms,splunk,New Relic, JIRA, BigPanda
Logging	Aws cloud watch Logs, S3 , Glacier, Aws log agent,fluentd

Lets start Scripting/Automation

Packer

```
{
  “variables”: {
        ---
   },
  “builders”: {
      ---
 },
 “provisioners”:[
  { type: file},
  {type: shell},
  {type: shell-local},
    ---
 ],
 “post-processors”:[
 ]
}
```

Ansible

```
roles/
    common/               # this hierarchy represents a "role"
        tasks/            #
            main.yml      #  <-- tasks file can include smaller files if warranted
        handlers/         #
            main.yml      #  <-- handlers file
        templates/        #  <-- files for use with the template resource
            ntp.conf.j2   #  <------- templates end in .j2
        files/            #
            bar.txt       #  <-- files for use with the copy resource
            foo.sh        #  <-- script files for use with the script resource
        vars/             #
            main.yml      #  <-- variables associated with this role
        defaults/         #
            main.yml      #  <-- default lower priority variables for this role
        meta/             #
            main.yml      #  <-- role dependencies

    webtier/              # same kind of structure as "common" was above, done for the webtier role
    monitoring/           # ""
    fooapp/               # ""
```

Examples for Practice :
https://github.com/raavula/devops

Terraform

```
$ tree sample-repo/terraform

terraform
├── GNUmakefile
├── README.md
├── modules
│   ├── ec2
│   │   ├── main.tf
│   │   ├── outputs.tf
│   │   └── variables.tf
│   └── vpc
│       ├── main.tf
│       ├── modules
│       │   ├── routing.tf
│       │   └── subnets.tf
│       ├── outputs.tf
│       └── variables.tf
└── roots
    ├── application1
    │   ├── env
    │   │   ├── production.tfvars
    │   │   ├── qa.tfvars
    │   │   └── staging.tfvars
    │   └── main.tf
    ├── application2
    │   ├── env
    │   │   ├── production.tfvars
    │   │   ├── qa.tfvars
    │   │   └── staging.tfvars
    │   └── main.tf
    └── vpc
        ├── env
        │   ├── production.tfvars
        │   ├── qa.tfvars
        │   └── staging.tfvars
        └── main.tf
```

Introduction to Docker

- Raghavaiah Avula

Traditional

<u>Advantages</u>

Utilization of raw resources

Isolation

<u>Disadvantages</u>

Slow deployment time

Expensive

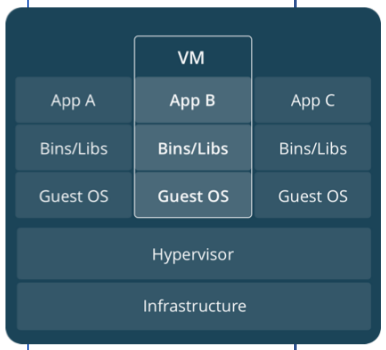
Wasted resources

Difficult to scale

Difficult to Migrate

Complex configuration

The Virtual Machine



<u>Advantages</u>

Good use of resources

Easy to scale

Easy to backup and migrate

Cost efficiency

Flexibility

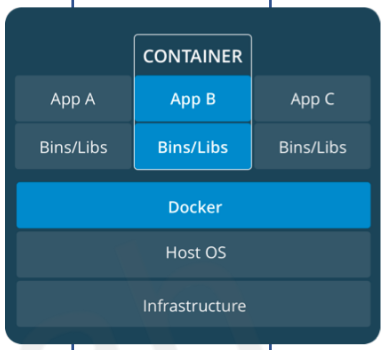
<u>Disadvantages</u>

Resource allocation is problematic

Vendor lock in

Complex configuration

Docker



<u>Advantages</u>

Isolation

Really Lightweight

Resource effective

Easy to migrate

Security

Low overhead

Mirror laptop to development to production

<u>Disadvantages</u>

Architecture

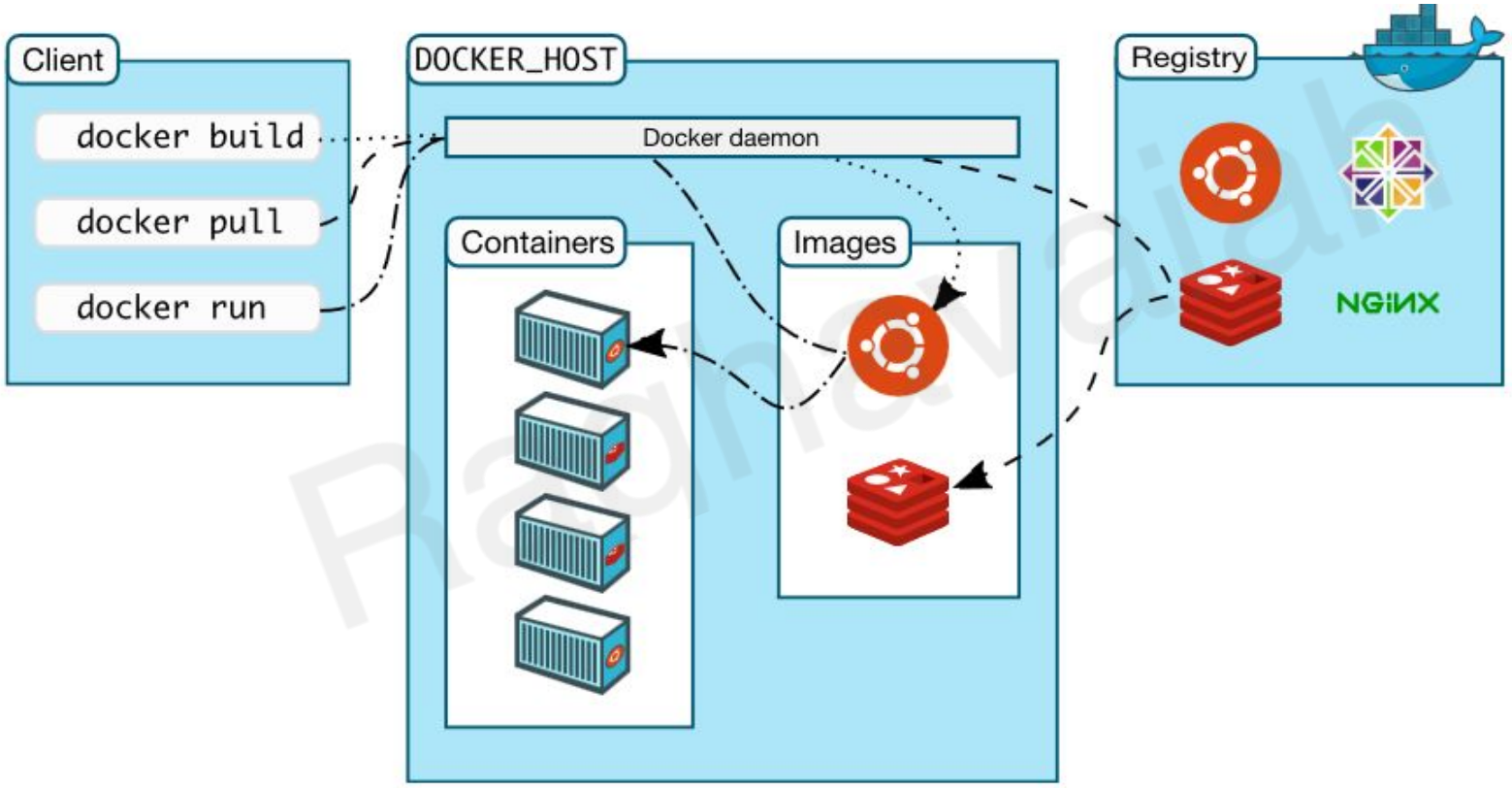
Security

Resource heavy apps

Docker’s core platform

- Docker Engine (Docker daemon)
- Docker Hub (Registry)
- Docker Machine (Host provisioning)
- Docker Swarm (Host clustering)
- Docker Compose (Container Orchestration)
- Docker Universal Control Plane (Container management/deployment)
- Kitematic (GUI)

Docker Architecture



Docker Concepts

- Get start with Docker
- Develop App in Docker
- Docker Networking
  - bridge
  - overlay
  - host
  - macvlan
- Manage Application data/storage
- Deploy in Production
  - logging
  - security
  - scale
  - limit resource
  - metrics
  - cgroups
  - user namespace
- Docker standards and compliance

Installing Docker

Follow below Link

https://docs.docker.com/glossary/?term=installation

cheat sheet

```
## List Docker CLI commands
docker
docker container --help

## Display Docker version and info
docker --version
docker version
docker info

## Execute Docker image
docker run hello-world

## List Docker images
docker image ls

## List Docker containers (running, all, all in quiet mode)
docker container ls
docker container ls --all
docker container ls -aq
```

Verify install

```
$ docker version
Client:
 Version:      1.9.0
 API version:  1.21
 Go version:   go1.4.3
 Git commit:   76d6bc9
 Built:        Tue Nov  3 19:20:09 UTC 2015
 OS/Arch:      darwin/amd64

Server:
 Version:      1.9.0
 API version:  1.21
 Go version:   go1.4.3
 Git commit:   76d6bc9
 Built:        Tue Nov  3 19:20:09 UTC 2015
 OS/Arch:      linux/amd64
```

First Container …

- Create an account in : https://hub.docker.com/
- Create first container
  git clone https://github.com/raavula/devops.git
  cd devops/docker/image

cheat sheet

```
docker build -t friendlyhello .  # Create image using this directory's Dockerfile
docker run -p 4000:80 friendlyhello  # Run "friendlyname" mapping port 4000 to 80
docker run -d -p 4000:80 friendlyhello         # Same thing, but in detached mode
docker container ls                                # List all running containers
docker container ls -a             # List all containers, even those not running
docker container stop <hash>           # Gracefully stop the specified container
docker container kill <hash>         # Force shutdown of the specified container
docker container rm <hash>        # Remove specified container from this machine
docker container rm $(docker container ls -a -q)         # Remove all containers
docker image ls -a                             # List all images on this machine
docker image rm <image id>            # Remove specified image from this machine
docker image rm $(docker image ls -a -q)   # Remove all images from this machine
docker login             # Log in this CLI session using your Docker credentials
docker tag <image> username/repository:tag  # Tag <image> for upload to registry
docker push username/repository:tag            # Upload tagged image to registry
docker run username/repository:tag                   # Run image from a registry
```

Docker Services ….

Examples:
https://github.com/raavula/devops/tree/master/docker/services

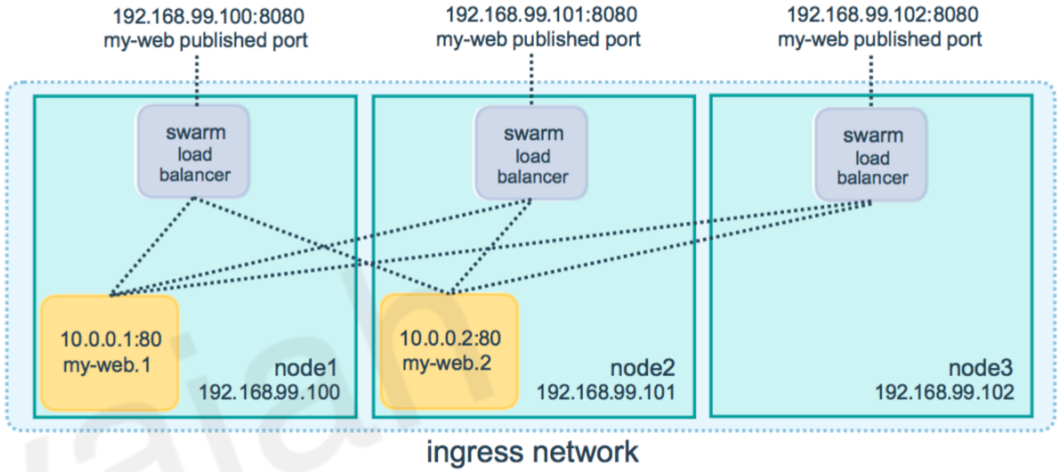
- Bring up the node in cluster
- Docker swarm init
- Create an service
- Create deployment

Cheat sheet

```
docker stack ls                                            # List stacks or apps
docker stack deploy -c <composefile> <appname>  # Run the specified Compose file
docker service ls                 # List running services associated with an app
docker service ps <service>                  # List tasks associated with an app
docker inspect <task or container>                   # Inspect task or container
docker container ls -q                                      # List container IDs
docker stack rm <appname>                                # Tear down an application
docker swarm leave --force      # Take down a single node swarm from the manager
```

Docker stack and Deployment

- Lets create an docker cluster
- Join worker to the cluster
- Bring up multiple services inside cluster
- Database service integration
- Docker service on cloud (aws/azure/digital ocean)



Cheat sheet

```
docker-machine create --driver virtualbox myvm1 # Create a VM (Mac, Win7, Linux)
docker-machine create -d hyperv --hyperv-virtual-switch "myswitch" myvm1 # Win10
docker-machine env myvm1                # View basic information about your node
docker-machine ssh myvm1 "docker node ls"         # List the nodes in your swarm
docker-machine ssh myvm1 "docker node inspect <node ID>"        # Inspect a node
docker-machine ssh myvm1 "docker swarm join-token -q worker"   # View join token
docker-machine ssh myvm1   # Open an SSH session with the VM; type "exit" to end
docker node ls                # View nodes in swarm (while logged on to manager)
docker-machine ssh myvm2 "docker swarm leave"  # Make the worker leave the swarm
docker-machine ssh myvm1 "docker swarm leave -f" # Make master leave, kill swarm
docker-machine ls # list VMs, asterisk shows which VM this shell is talking to
docker-machine start myvm1            # Start a VM that is currently not running
docker-machine env myvm1      # show environment variables and command for myvm1
eval $(docker-machine env myvm1)         # Mac command to connect shell to myvm1
& "C:\Program Files\Docker\Docker\Resources\bin\docker-machine.exe" env myvm1 | Invoke-Expression   # Windows
docker stack deploy -c <file> <app>  # Deploy an app; command shell must be set to talk to manager (myvm1), us
docker-machine scp docker-compose.yml myvm1:~ # Copy file to node's home dir (only required if you use ssh to
docker-machine ssh myvm1 "docker stack deploy -c <file> <app>"   # Deploy an app using ssh (you must have firs
eval $(docker-machine env -u)     # Disconnect shell from VMs, use native docker
docker-machine stop $(docker-machine ls -q)               # Stop all running VMs
docker-machine rm $(docker-machine ls -q) # Delete all VMs and their disk images
```

Introduction to K8s

- Raghavaiah Avula

K8s Concepts

- K8s and container operations
- K8s core concepts and constructs
- Network, storage and services
- Updates and gradual rollouts
- Continuous Delivery
- Monitoring and Logging
- Standards and specifications
- Production Ready

Why k8s ?

Some container orchestration tools to know about include:

Amazon ECS -- The Amazon EC2 Container Service (ECS) supports Docker containers and lets you run applications on a managed cluster of Amazon EC2 instances.

Azure Container Service (ACS) -- ACS lets you create a cluster of virtual machines that act as container hosts along with master machines that are used to manage your application containers.

Cloud Foundry's Diego -- Diego is a container management system that combines a scheduler, runner, and health manager. It is a rewrite of the Cloud Foundry runtime.

CoreOS Fleet -- Fleet is a container management tool that lets you deploy Docker containers on hosts in a cluster as well as distribute services across a cluster.

Docker Swarm -- Docker Swarm provides native clustering functionality for Docker containers, which lets you turn a group of Docker engines into a single, virtual Docker engine.

Google Container Engine -- Google Container Engine, which is built on Kubernetes, lets you rur Docker containers on the Google Cloud platform. It schedules containers into the cluster and manages them based on user-defined requirements.

Kubernetes -- Kubernetes is an orchestration system for Docker containers. It handles scheduling and manages workloads based on user-defined parameters.

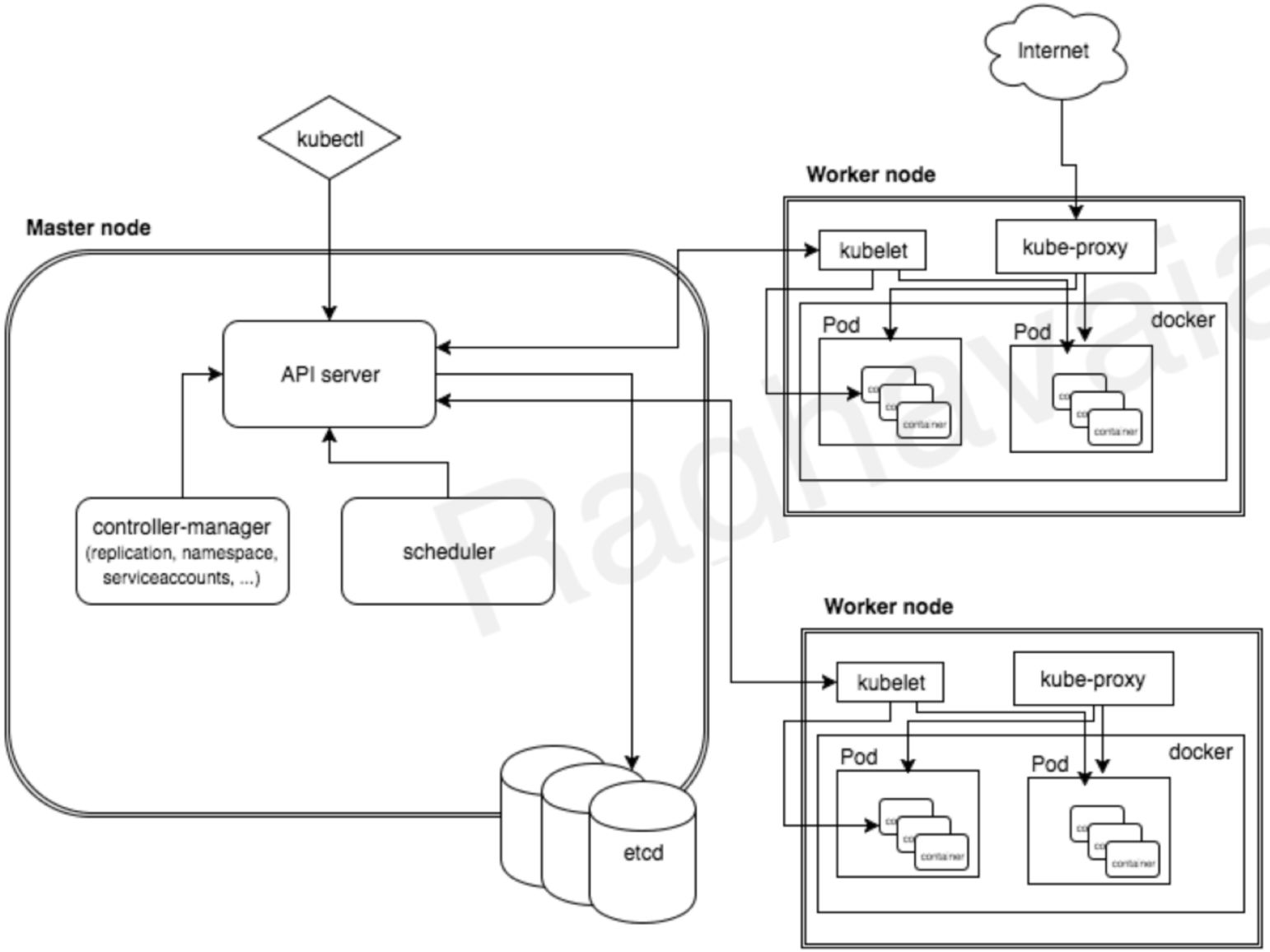
Mesosphere Marathon -- Marathon is a container orchestration framework for Apache Mesos that is designed to launch long-running applications. It offers key features for running applications in a clustered environment.

Additionally, the Cloud Native Computing Foundation (CNCF) is focused on integrating the orchestration layer of the container ecosystem. The CNCF's stated goal is to create and drive adoption of a new set of common container technologies, and it recently selected Google's Kubernetes container orchestration tool as its first containerization technology.

Orchestration manages

- Container scheduling
  - Distributing containers to appropriate hos
  - Host resource leveling
  - Availability zone diversity
  - Related container packaging and co-deployment
- Container management
  - Monitoring and recovery
  - Image upgrade rollout
  - Scaling
  - Logging
- Service Endpoints
  - Discovery
  - HA
  - Load Balancing
  - Auto scaling
- External service
  - Network management
  - volume management

Kubernetes Architecture



Master Nodes

API server
Scheduler
Controller-manager
Kubectl
etcd

Worker Nodes

Kubelet
Kube-proxy
docker
pod

Network

ACI
AOS
Big Cloud Fabric
Cilium
CNI-Genie
Contiv
Contrail
Flannel
GCE
Knitter
kube-router
OpenVSwitch
OVN
Project Calico
Romana
Weave

CI/CD

- Gulp.js
- Jenkin plugin(kubernetes)

Monitoring

- sysdig
- twistlock

Metrics

- Heapster
- cAdvisor
- Contrib
- InfluxDB

services

- Internal service (cluster IP)
- External service
  - Load balancer
  - Node Port

UI

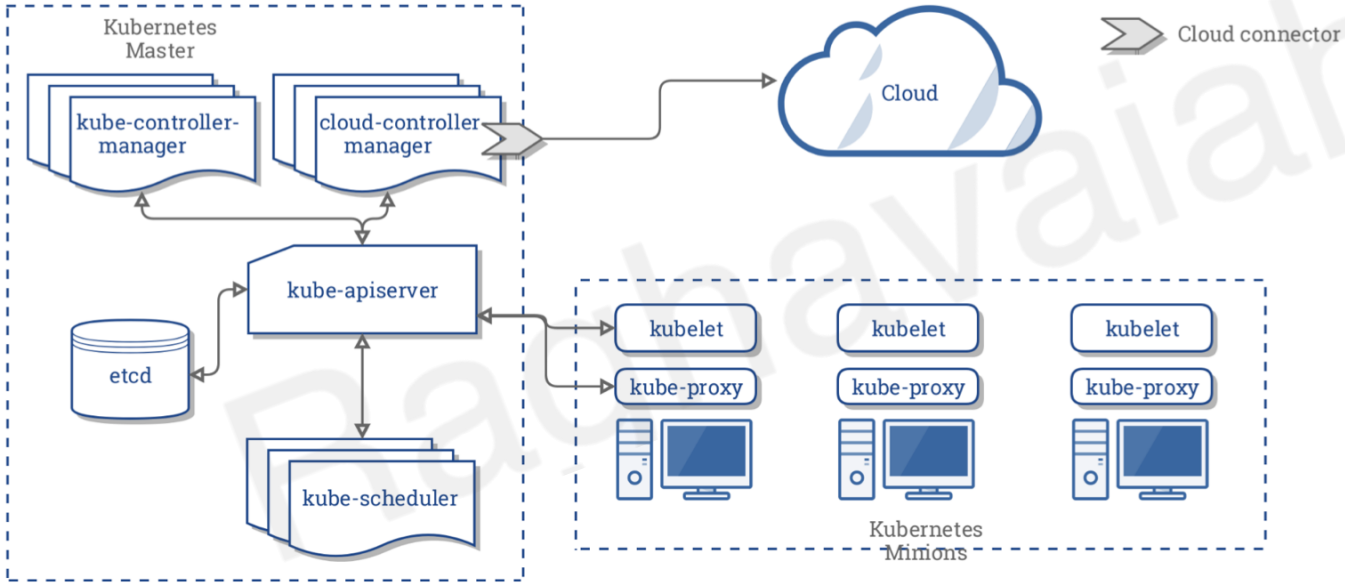
- kubernetes dashboard UI
- Grafana (metrics on the cluster)
- swagger (interactions and easy discovery of API)

Logging

FluentD

Standards

OCI
CNCF
CoreOS
Tectonics



concepts

- pods
- labels
- Replication controllers (RC)
- Health checks (http,tcp,command)
- Service discovery (environment variable or DNS)
- multi port support
- multitenancy with namespaces
- resource limit and quotas
- scale and rolling-update

persistence volumes

- temporary disk
- cloud volumes like
  - GCE persistent volume
  - AWS Ebs
  - NFS

Network

- Docker Engine (bridge network)
- libenetwork
- weave
- Flannel
- project calico

Install k8s and Hands on

Install:
https://kubernetes.io/docs/setup/pick-right-solution/#local-machine-solutions

Examples:
https://github.com/raavula/devops/tree/master/k8s

kube-dashboard = k8s UI access
guestbook = k8s stateless application example
wordpress = k8s statefull application example

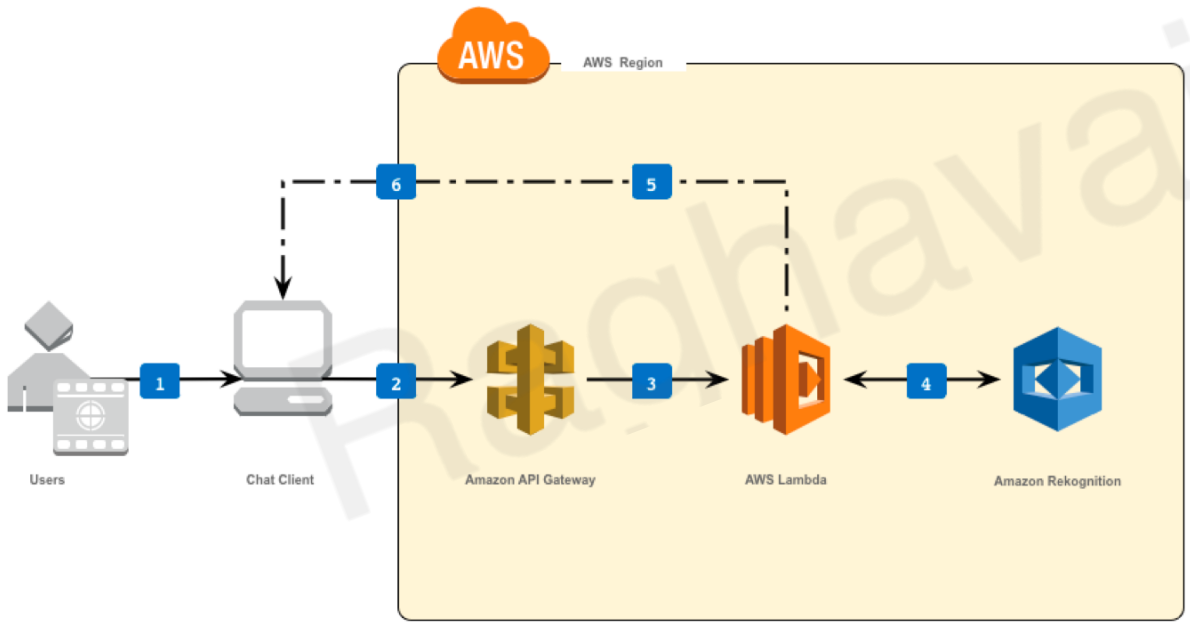
AWS Reference Architectures

Serverless Reference Architecture: Image Moderation Chatbot

Image Moderation Chatbot

Building a Chatbot on AWS

This reference architecture shows you how to build a serverless chatbot on AWS that monitors your chat channels and removes images containing suggestive or explicit content.



1. A user posts a message containing an image to a chat app channel that's monitored by a chatbot.
2. The chat app posts the event to an **Amazon API Gateway** API for the chatbot.
3. The chatbot validates the event. This event triggers an **AWS Lambda** function that downloads the image.
4. **Amazon Rekognition**'s image recognition feature checks the image for suggestive or explicit content.
5. The chat app API deletes an image containing explicit or suggestive content from the chat channel.
6. The chatbot uses the chat app API to post a message to the chat channel detailing deletion of the image.

amazon web services

AWS Reference Architectures

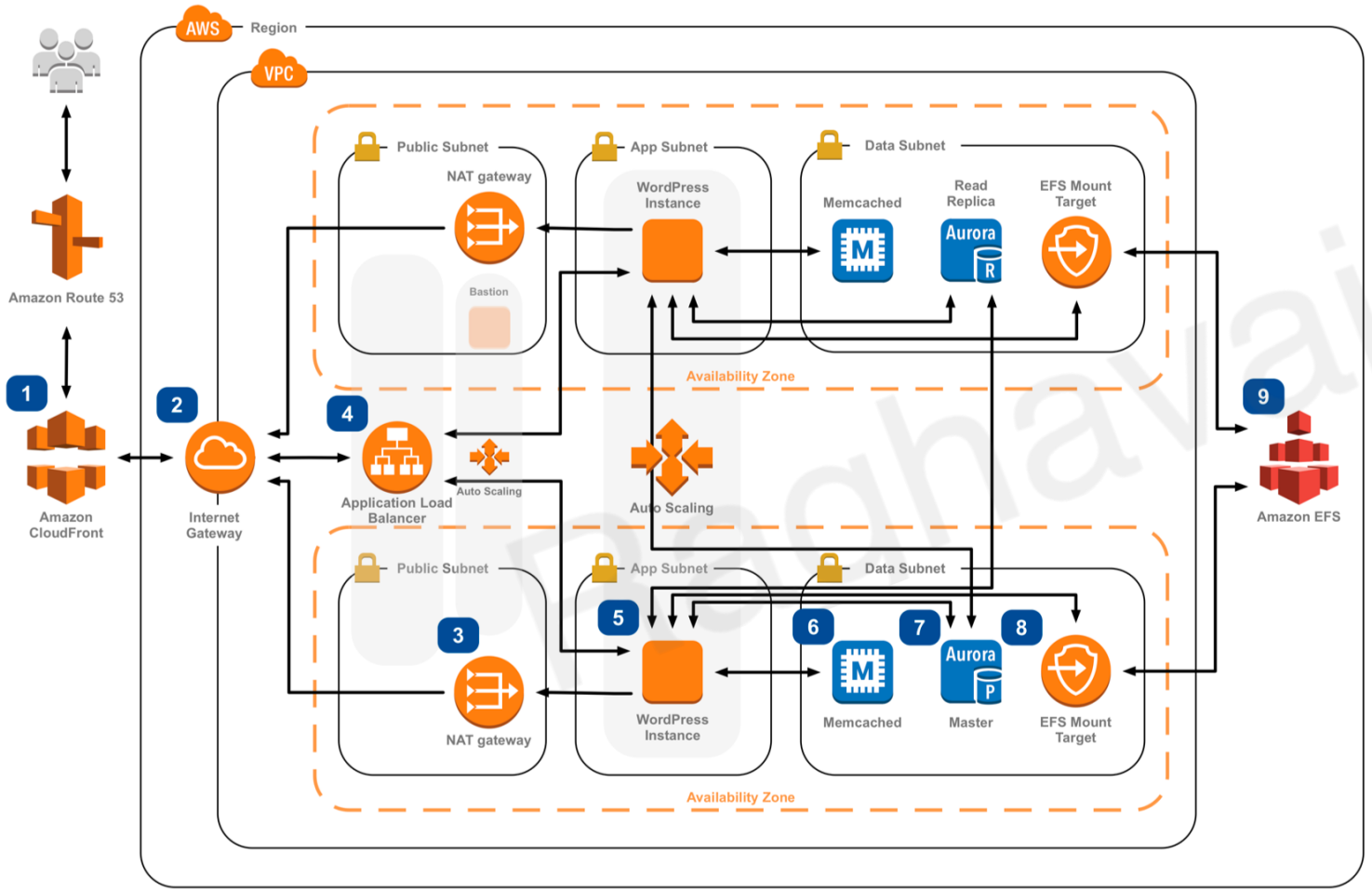
Reference Link

https://github.com/aws-samples/lambda-refarch-image-moderation-chatbot

WordPress Hosting

How to run WordPress on AWS

WordPress is one of the world's most popular web publishing platforms, being used to publish 27% of all websites, from personal blogs to some of the biggest news sites. This reference architecture simplifies the complexity of deploying a scalable and highly available WordPress site on AWS.



1. Static and dynamic content is delivered by **Amazon CloudFront**.
2. An **Internet gateway** allows communication between instances in your VPC and the Internet.
3. **NAT gateways** in each public subnet enable Amazon EC2 instances in private subnets (App & Data) to access the Internet.
4. Use an **Application Load Balancer** to distribute web traffic across an Auto Scaling Group of Amazon EC2 instances in multiple AZs.
5. Run your WordPress site using an **Auto Scaling group** of **Amazon EC2 instances**. Install the latest versions of WordPress, Apache web server, PHP 7, and OPcache and build an Amazon Machine Image that will be used by the Auto Scaling group launch configuration to launch new instances in the Auto Scaling group.
6. If database access patterns are read-heavy, consider using a WordPress plugin that takes advantage of a caching layer like **Amazon ElastiCache** (**Memcached**) in front of the database layer to cache frequently accessed data.
7. Simplify your database administration by running your database layer in **Amazon RDS** using either Aurora or MySQL.
8. Amazon EC2 instances access shared WordPress data in an Amazon EFS file system using **Mount Targets** in each AZ in your VPC.
9. Use **Amazon EFS**, a simple, highly available, and scalable network file system so WordPress instances have access to your shared, unstructured WordPress data, like php files, config, themes, plugins, etc.

amazon web services

AWS Reference Architectures

Tenant isolation

- account level
- vpc level
- subnet level
- container layer
- application level with tenant ID in same DB

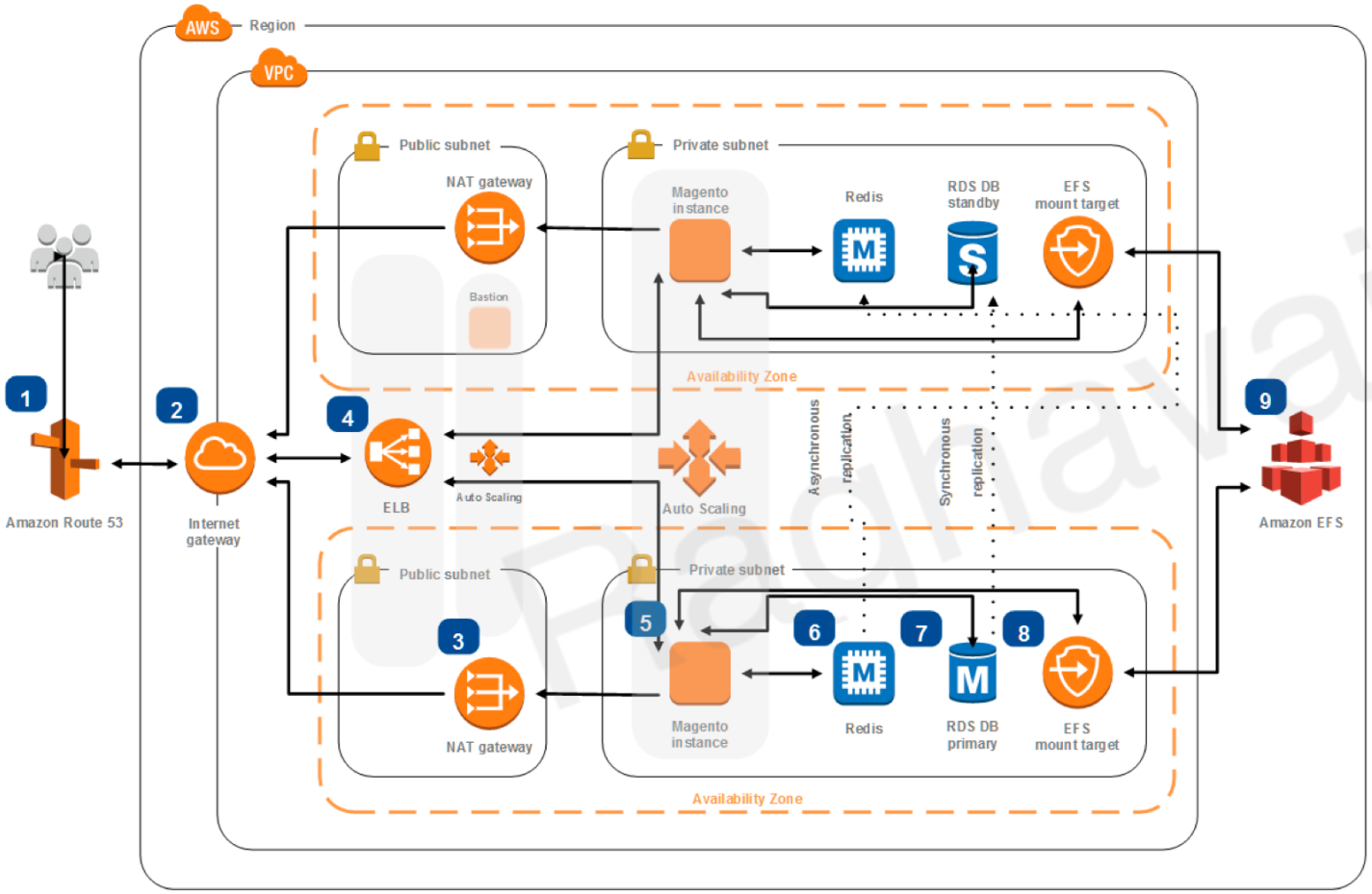
Reference

https://github.com/aws-samples/aws-refarch-wordpress

Magento CE Hosting

Running Magento Community Edition (CE) on AWS

Magento Community Edition (CE) is a flexible, open-source commerce platform for developers and small businesses. This reference architecture simplifies the complexity of deploying a scalable and highly available Magento CE commerce platform on AWS.



1. **Amazon Route 53** provides DNS configuration and routes traffic to Elastic Load Balancing (ELB) endpoints.
2. An **internet gateway** allows communication between instances in your VPC and the internet.
3. **NAT gateways** in each public subnet enable Amazon EC2 instances in private subnets to access the internet.
4. Use an **ELB Load Balancer** to distribute web traffic across an Auto Scaling group of Amazon EC2 instances in multiple Availability Zones.
5. Run your Magento commerce site using an **Auto Scaling group** of **Amazon EC2 instances**. Install the latest versions of Magento CE, Nginx web server, and PHP 7. Then, build an Amazon Machine Image (AMI) that the Auto Scaling group launch configuration can use to launch new instances in the group.
6. If database access patterns are read-heavy, consider using a caching layer like **Amazon ElastiCache for Redis** in front of the database layer to cache frequently accessed data.
7. Simplify your database administration by running your database layer in **Amazon RDS** using either Aurora or MySQL.
8. Amazon EC2 instances access the shared Magento data in an Amazon EFS file system using **mount targets** in each Availability Zone in your VPC.
9. Use an **Amazon EFS** network file system so that Magento instances can access your shared, unstructured Magento data such as images, media files etc.

amazon web services

AWS Reference Architectures

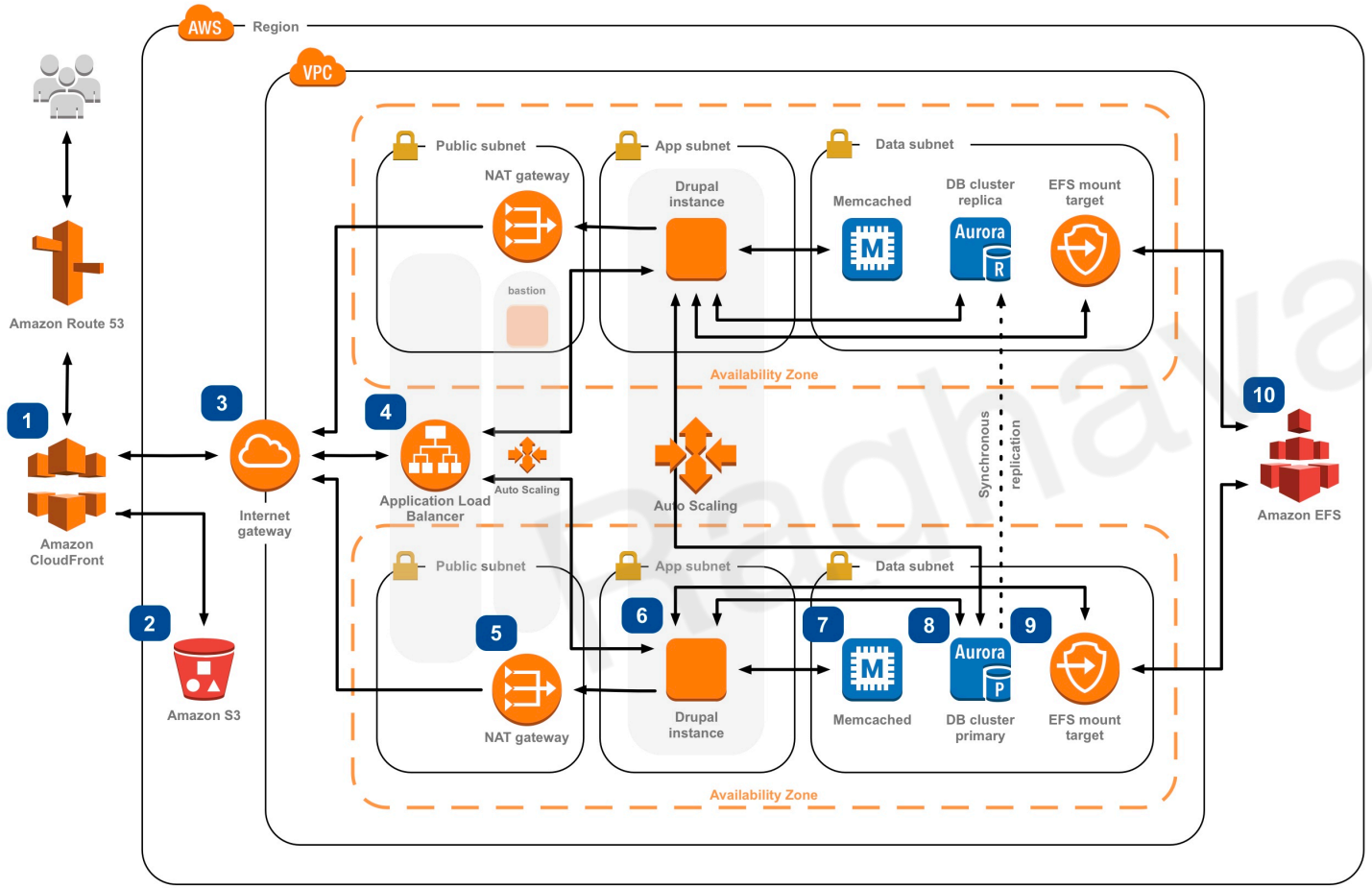
Reference

https://github.com/amazon-archives/aws-refarch-magento

Drupal Hosting

Running Drupal on AWS

Drupal is a free, open-source web content management platform. The Drupal community is one of the largest open-source communities in the world with more than 1,000,000 passionate developers, designers, trainers, strategists, coordinators, editors, and sponsors. This reference architecture enables you to deploy a scalable and highly available Drupal site on AWS.



1. Use **Amazon CloudFront** to deliver static and dynamic content.
2. Use **Amazon S3** to store static content such as media, downloadable files, etc.
3. Attach an **internet gateway** to your VPC to enable communication between Amazon EC2 instances in your VPC and the internet.
4. Use an **Application Load Balancer** to distribute web traffic across an Auto Scaling Group of Amazon EC2 instances in multiple Availability Zones.
5. Create a **NAT gateway** in each public subnet to send traffic to the internet gateway from private subnets (app and data) in your VPC.
6. Run your Drupal site using an **Auto Scaling group** of **Amazon EC2 instances**. Install the latest versions of Drupal, Apache web server, PHP 7, and OPcache. Then, build an Amazon Machine Image (AMI) that the Auto Scaling group launch configuration can use to launch new instances in the group.
7. If database access patterns are read-heavy, consider using a Drupal plugin that takes advantage of a caching layer such as **Amazon ElastiCache for Memcached** in front of the database layer to cache frequently accessed data.
8. Run your database layer in **Amazon RDS** using either Aurora or MySQL to simplify your database administration.
9. After creating an Amazon EFS file system, create **mount targets.** Mount the file system on your Drupal Amazon EC2 instances in each Availability Zone in your VPC.
10. Use **Amazon EFS** for Drupal instances to access your shared, unstructured Drupal data such as PHP files, config, themes, plugins, etc.

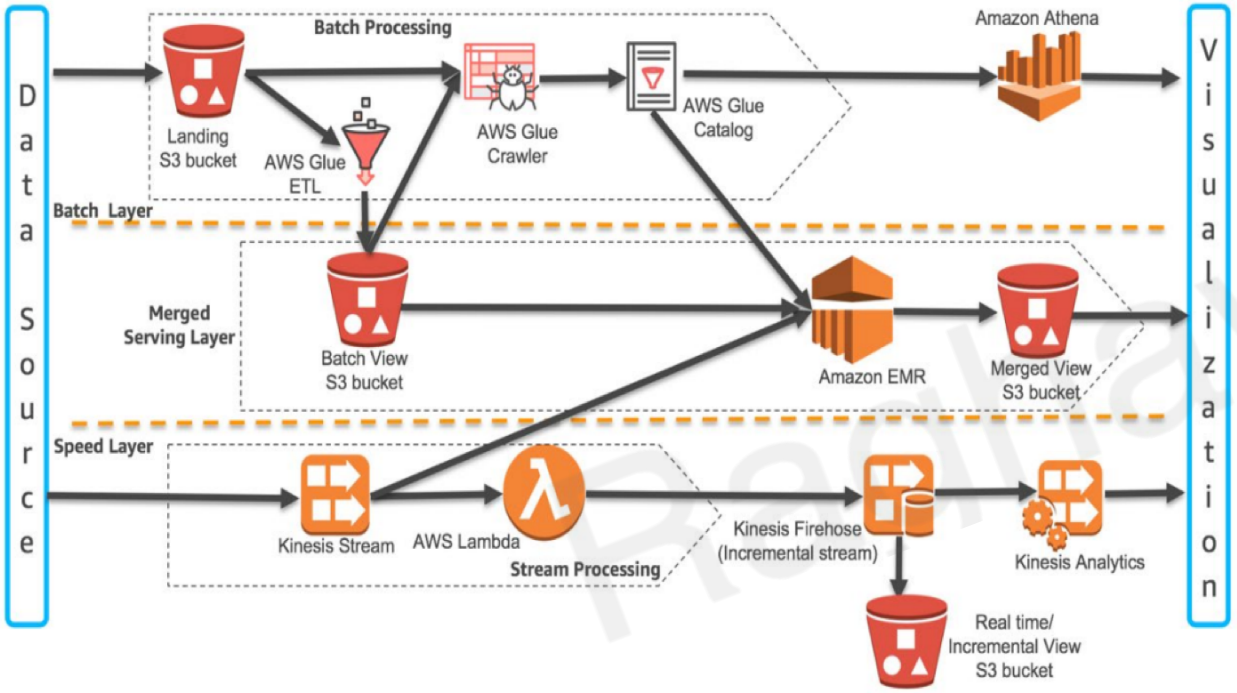
amazon web services

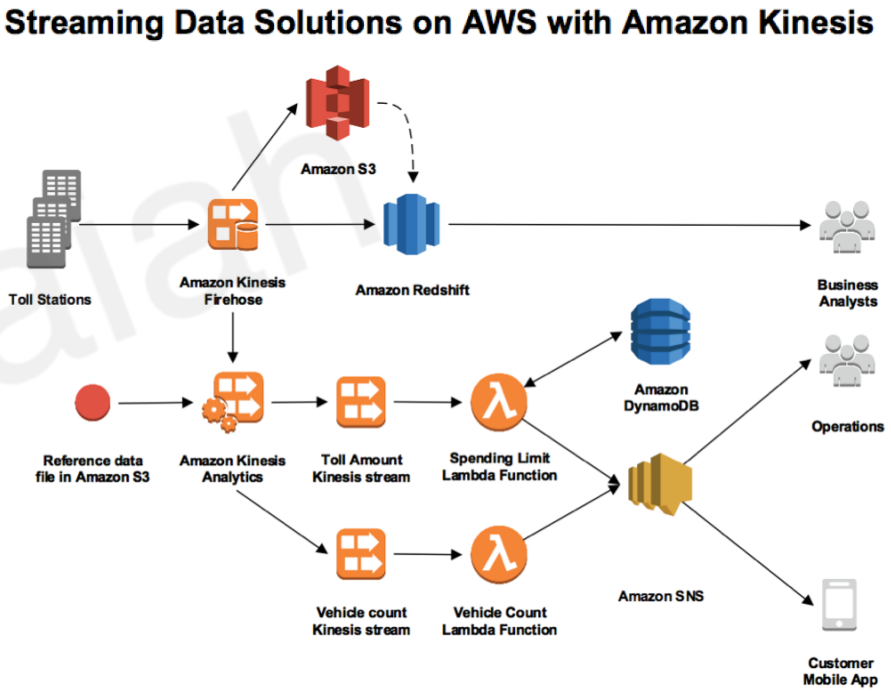
AWS Reference Architectures

Reference

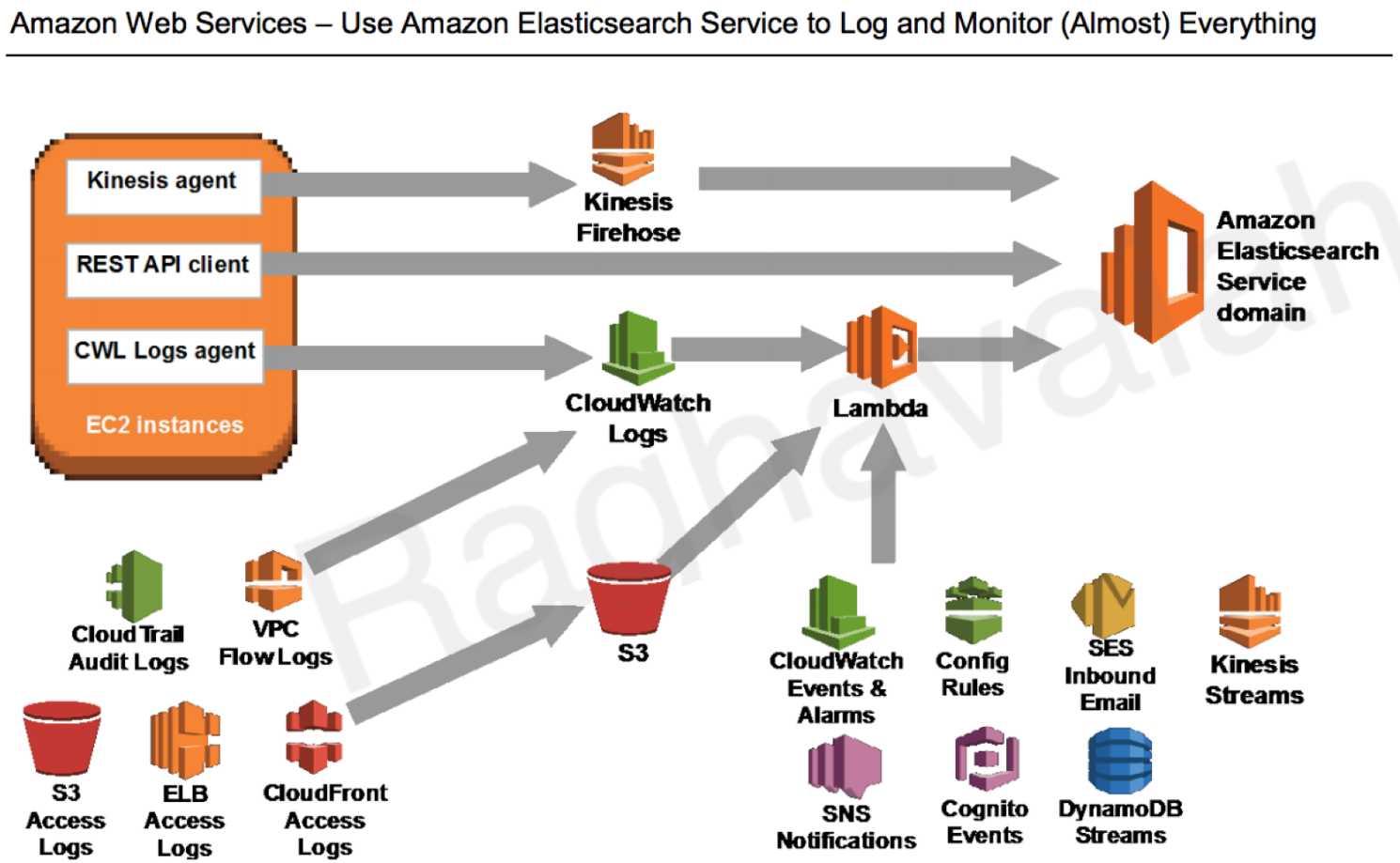
https://github.com/aws-samples/aws-refarch-drupal

Streaming,Big Data & Analytics





Logs and Monitoring Pipeline



Amazon Web Services – **Building Media & Entertainment Predictive Analytics Solutions**