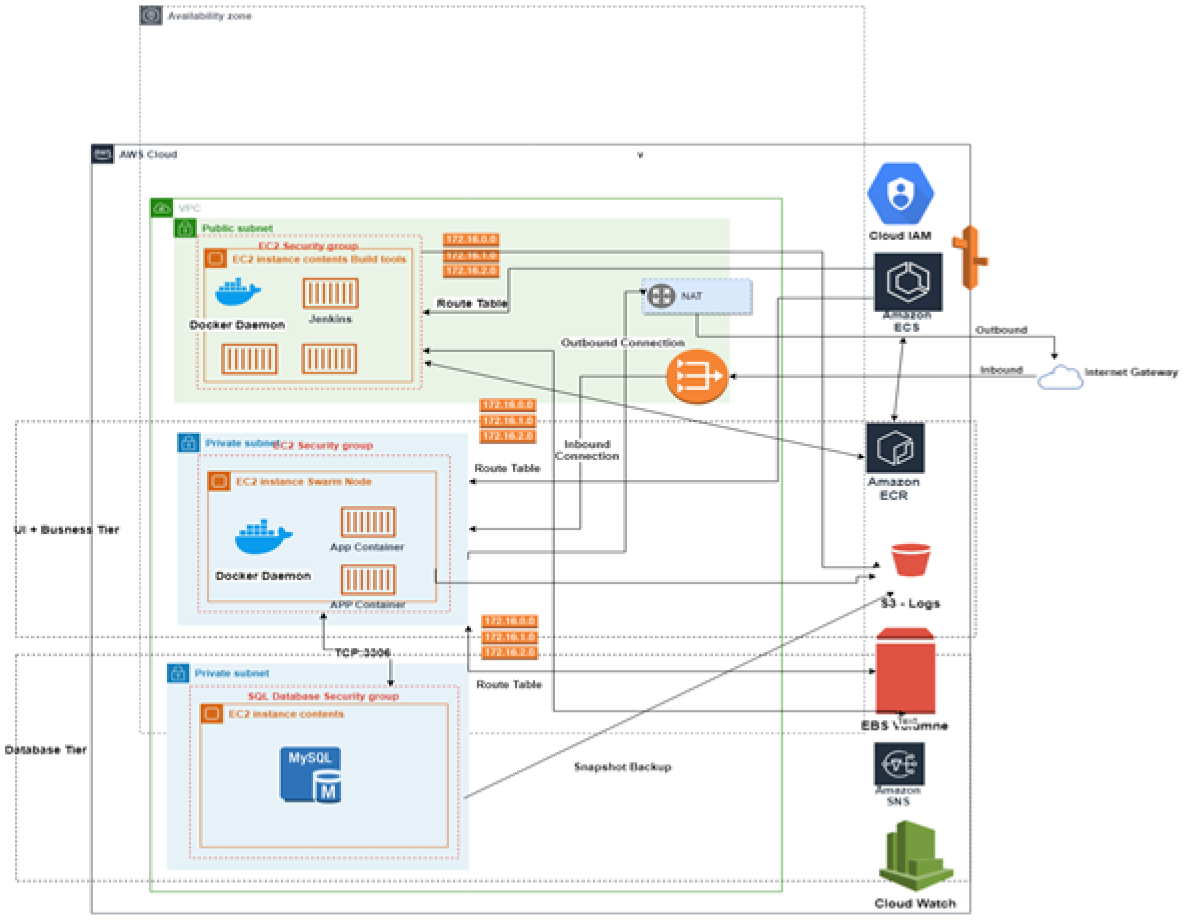
Deployment Architecture

Deployment Architecture for Simulator

For initial generic understanding please refer Overview of AWS ECS Container . Below is the implementation infrastructure layout for the Simulator.



Above diagram is PCI Compliance since it has App and data layer is hidden in the private subnet and thus avoiding direct access to it.

· Route 53: DNS service cluster that allows an Amazon Web Services customer to define how end-user traffic is routed to application endpoints through a visual interface.

· Virtual Private Cloud: An ECS cluster runs within a VPC.

Here we have 1 Public and 2 Private Subnets.

· Public Subnet: It will have the EC2 instance contents the build tools container like Jenkins, sonarqube etc.

· App Private Subnet – It will have EC2 instance contents docker container with **required environment namespace (Prod, QA and Dev)**. Container will have app component running into it.

· Data Private Subnet: AWS SQL RDS managed database service.

· Elastic Container Registry: Simulator application images are stored.

· Elastic Block Store: This service provides persistent block storage for ECS tasks (workloads running in containers).

· AWS S3: File storage used to store logs, database snapshot back up and CloudFormation scripts.

· Amazon ECS: Managed Service is the Docker-compatible container orchestration solution from Amazon Web Services. It allows you to run containerized applications on EC2 instances and scale both of them. The following diagram shows the high-level.

· Elastic Load Balancer: The Network Load balancer is used to direct traffic to containers. Visibility - It will be in Public Subnet.

· CloudWatch: Distributed log collection tool used to view the status of containers.

· CloudTrail: This service can log ECS API calls. Details captured include type of request made to Amazon ECS, source IP address, user details, etc.

· AWS System Manager - Parameter Store: A Secure password storage mechanism, which is used to store the database password. Running the sub-generator will introduce a new Spring Cloud component which will read in the password on application startup.

· AWS SES: Email Server needed to send the email to the end user.

· AWS SNS: Email Notification Service for operation side notification.

· AWS - IAM Role: Use to create a new role for developer, QA and DevOps which the ECS tasks will execute under, with an associated policy.

· AWS Code Commit: Used for source code repository.

· AWS NAT: use a network address translation (***NAT***) instance in a public subnet in your VPC to enable instances in the private subnet to initiate outbound IPv4 traffic to the Internet. Thus EC2 instances in the App private subnet can access the Internet by using a network address translation (NAT) gateway that resides in the public subnet. The database servers can connect to the Internet for software updates using the NAT gateway, but the Internet cannot establish connections to the database servers.

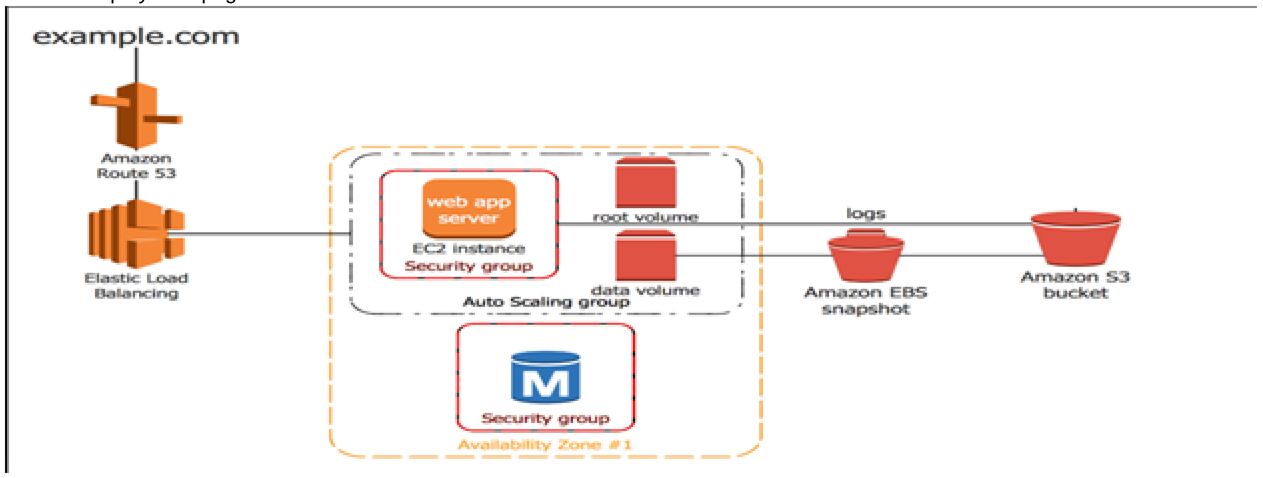
· AWS Cloudformation: All required services (besides AWS System Manager Parameters) are defined in a set of CloudFormation files. The base file contains high level services, and then each application is defined in its own file, which is called a nested stack.

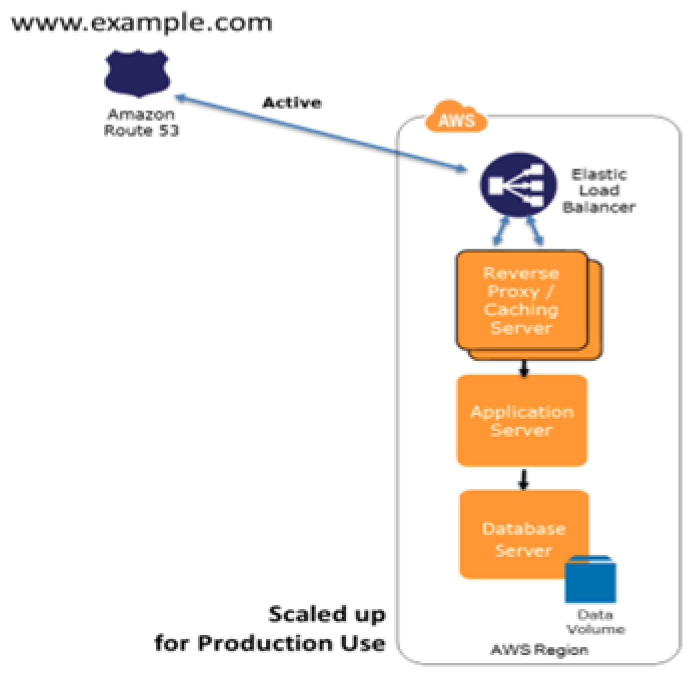
· AWS Managed RDS: Using RDS managed services we can create as needed database instances.We will have separate databases – i.e. MC-Prod, MC-Stg, MC-Dev using RDS service

Amazon RDS creates and saves automated backups of your DB instance. Amazon RDS creates a storage volume snapshot of your DB instance, backing up the entire DB instance and not just individual databases.

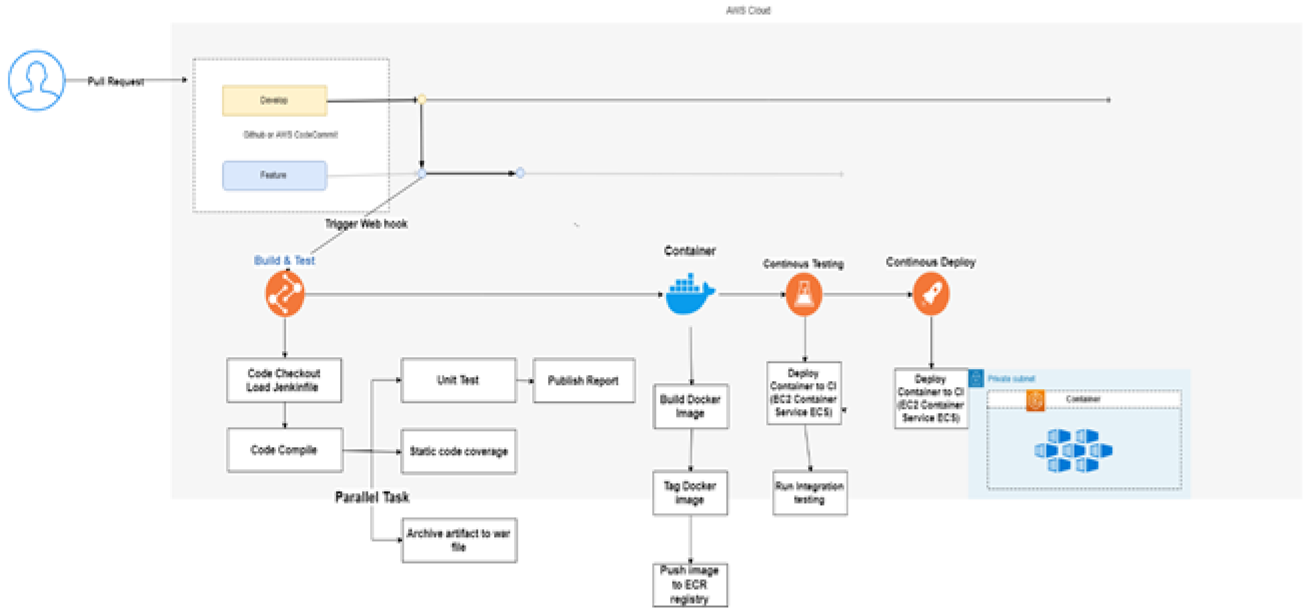
AWS Route 53 Flow

1. A user opens a web browser, enters www.example.com in the address bar, and presses Enter.
2. The request for www.example.com is routed to a DNS resolver, which is typically managed by the user's Internet service provider (ISP), such as a cable Internet provider, a DSL broadband provider, or a corporate network.
3. The DNS resolver for the ISP forwards the request for www.example.com to a DNS root name server.
4. The DNS resolver for the ISP forwards the request for www.example.com again, this time to one of the TLD name servers for .com domains. The name server for .com domains responds to the request with the names of the four Amazon Route 53 name servers that are associated with the example.com domain.
5. The DNS resolver for the ISP chooses an Amazon Route 53 name server and forwards the request for www.example.com to that name server.
6. The Amazon Route 53 name server looks in the example.com hosted zone for the www.example.com record, gets the associated value, such as the IP address for a web server, 192.0.2.44, and returns the IP address to the DNS resolver.
7. The DNS resolver for the ISP finally has the IP address that the user needs. The resolver returns that value to the web browser. The DNS resolver also caches (stores) the IP address for example.com for an amount of time that you specify so that it can respond more quickly the next time someone browses to example.com. For more information, see time to live (TTL).
8. The web browser sends a request for www.example.com to the IP address that it got from the DNS resolver. This is where your content is, for example, a web server running on an Amazon EC2 instance or an Amazon S3 bucket that's configured as a website endpoint.
9. The web server or other resource at 192.0.2.44 returns the web page for www.example.com to the web browser, and the web browser displays the page.





Container Flow Diagram



On every push to the feature branch, pipeline is triggered. The Pipeline starts with:

- Checkout of source code
- Compiling the code
- Creating a docker image
- Tagging the image.

Newly built image is pushed to the artifactory docker registry.

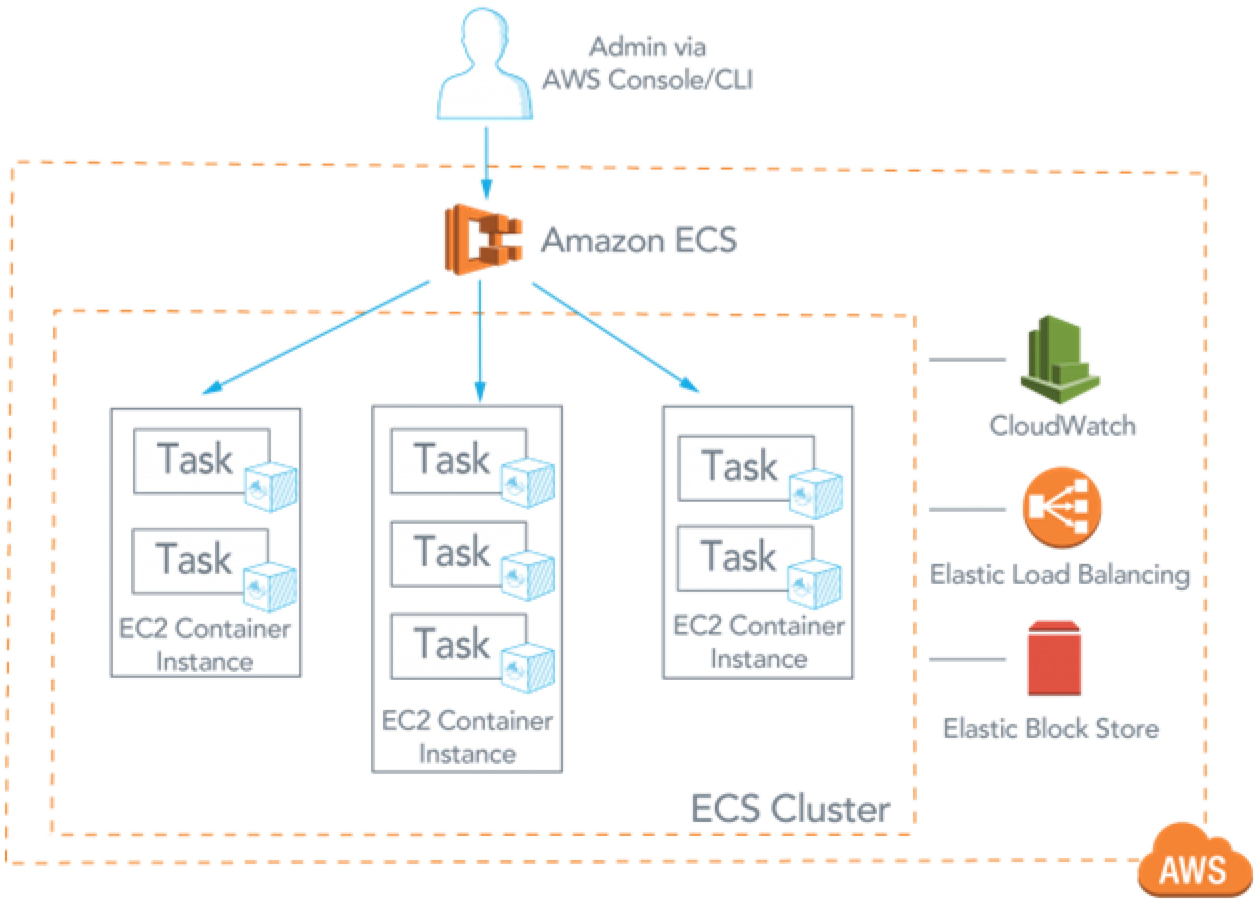
ECS will deploy latest image from the ECR to the container.

Git Branch Strategy

Branch	Purpose	Details
FEATURE	Unit of change for a Story scheduled for an upcoming release	Developer create a feature branch Pushes often to feature branch. CI builds triggered on push to branch Deploy to DEV environment. Container based apps. Automatically deploy nightly. Rolling update de
DEVELOP	Integration Branch Reflect code for next scheduled release	Long Living branch. Dev leads merge to protected branch. Builds triggered on merge Deployed to protect DEV-INT environment.
RELEASE	Release Candidate: Used for preparing and vetting a release before production	Short living branch that is merged to master and deleted when release to production. Cut from develop branch when Dev Lead determines there is sufficient content for the next release. -Manual one
MASTER	Reflects code currently in Production	Used for record keeping. It contains the Stable code.
HOTFiX	Production Fixes	Branched from a release tag. Similar to release branch. Fix is vetted on this branch prior to release back to production. Merged to Master and develop branch.

Overview of Amazon ECS container orchestration

Amazon ECS is the Docker-compatible container orchestration solution from Amazon Web Services. It allows you to run containerized applications on EC2 instances and scale both of them. The following diagram shows the high-level architecture of ECS.



As shown above, ECS Clusters consist of tasks which run in Docker containers, and container instances, among many other components. Here are some AWS services commonly used with ECS:

Elastic Load Balancer: This component can route traffic to containers. Two kinds of load balancing are available: application and classic.

Elastic Block Store: This service provides persistent block storage for ECS tasks (workloads running in containers).

Cloud Watch: This service collects metrics from ECS. Based on Cloud Watch metrics, ECS services can be scaled up or down.

Virtual Private Cloud: An ECS cluster runs within a VPC. A VPC can have one or more subnets.

Cloud Trail: This service can log ECS API calls. Details captured include type of request made to Amazon ECS, source IP address, user details, etc.

ECS, which is provided by Amazon as a service, is composed of multiple built-in components which enable administrators to create clusters, tasks and services:

State Engine: A container environment can consist of many EC2 container instances and containers. With hundreds or thousands of containers, it is necessary to keep track of the availability of instances to serve new requests based on CPU, memory, load balancing, and other characteristics. The state engine is designed to keep track of available hosts, running containers, and other functions of a cluster manager.

Schedulers: These components use information from the state engine to place containers in the optimal EC2 container instances. The batch job scheduler is used for tasks that run for a short period of time. The service scheduler is used for long running apps. It can automatically schedule new tasks to an ELB.

Cluster: This is a logical placement boundary for a set of EC2 container instances within an AWS region. A cluster can span multiple availability zones (AZs), and can be scaled up/down dynamically. Dev/Test environment may have 2 clusters: 1 each for production and test.

Tasks: A task is a unit of work. Task definitions, written in JSON, specify containers that should be co-located (on an EC2 container instance). Though tasks usually consist of a single container, they can also contain multiple containers.

Services: This component specifies how many tasks should be running across a given cluster. You can interact with services using their API, and use the service scheduler for task placement.

Note that ECS only manages ECS container workloads – resulting in vendor lock-in. There's no support to run containers on infrastructure outside of EC2, including physical infrastructure or other clouds such as Google Cloud Platform and Microsoft Azure. The advantage, of course, is the ability to work with all the other AWS services like Elastic Load Balancers, Cloud Trail, and Cloud Watch etc.