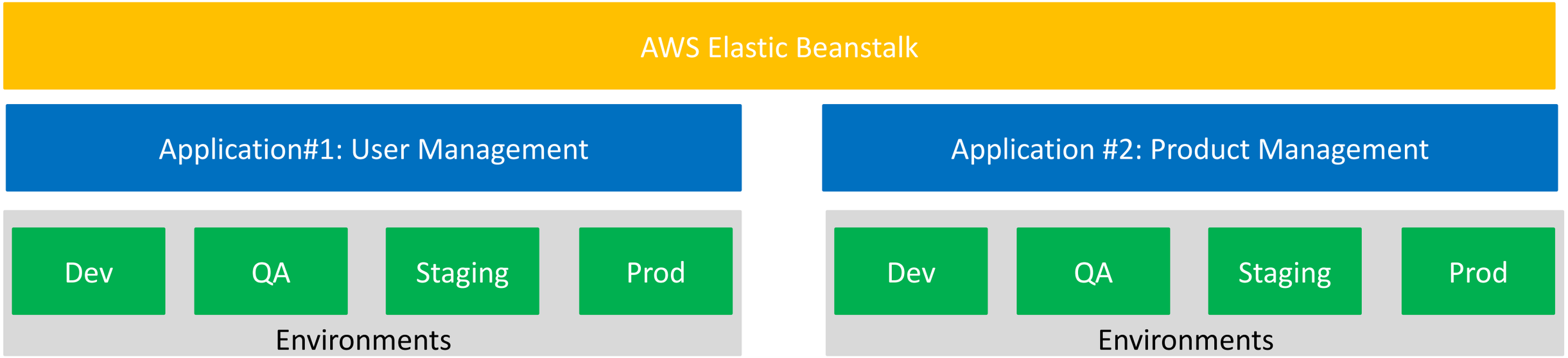
AWS Elastic Beanstalk Masterclass

Kalyan Reddy Daida

AWS Elastic Beanstalk Masterclass

Course Contents

Elastic Beanstalk – First Steps



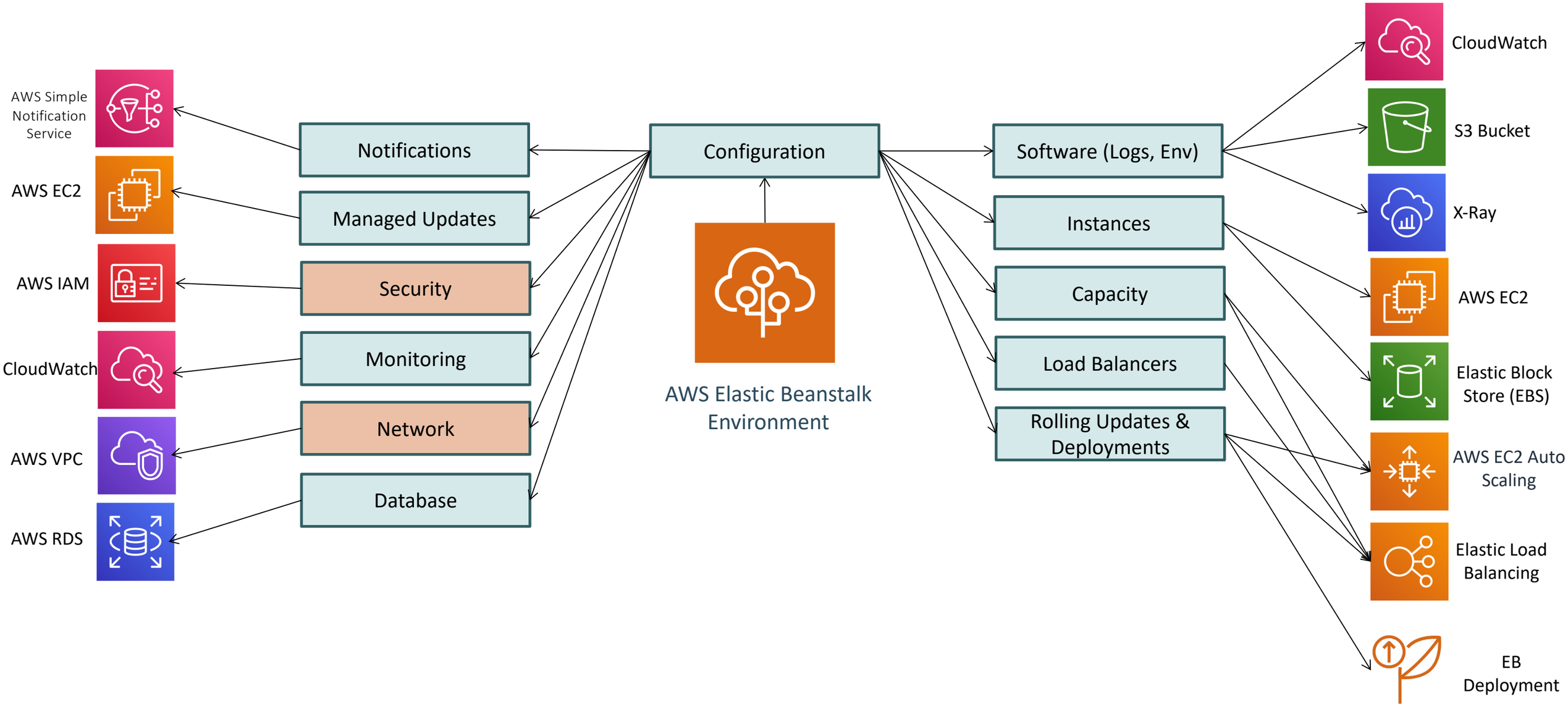
EB Environment Features

AWS Elastic Beanstalk Environment

- Dashboard
- Configuration
- Logs
- Health
- Monitoring
- Alarms
- Managed Updates
- Events
- Tags
- Actions

- Save Configuration
- Load Configuration
- Swap Environment URLs
- Clone Environment
- Clone with Latest Platform
- Abort Current Operation
- Restart App Server(s)
- Rebuild Environment
- Terminate Environment
- Restore Terminated Environment (From Applications screen)

Elastic Beanstalk Environment - Configuration



AWS Elastic Beanstalk

Application#1: User Management

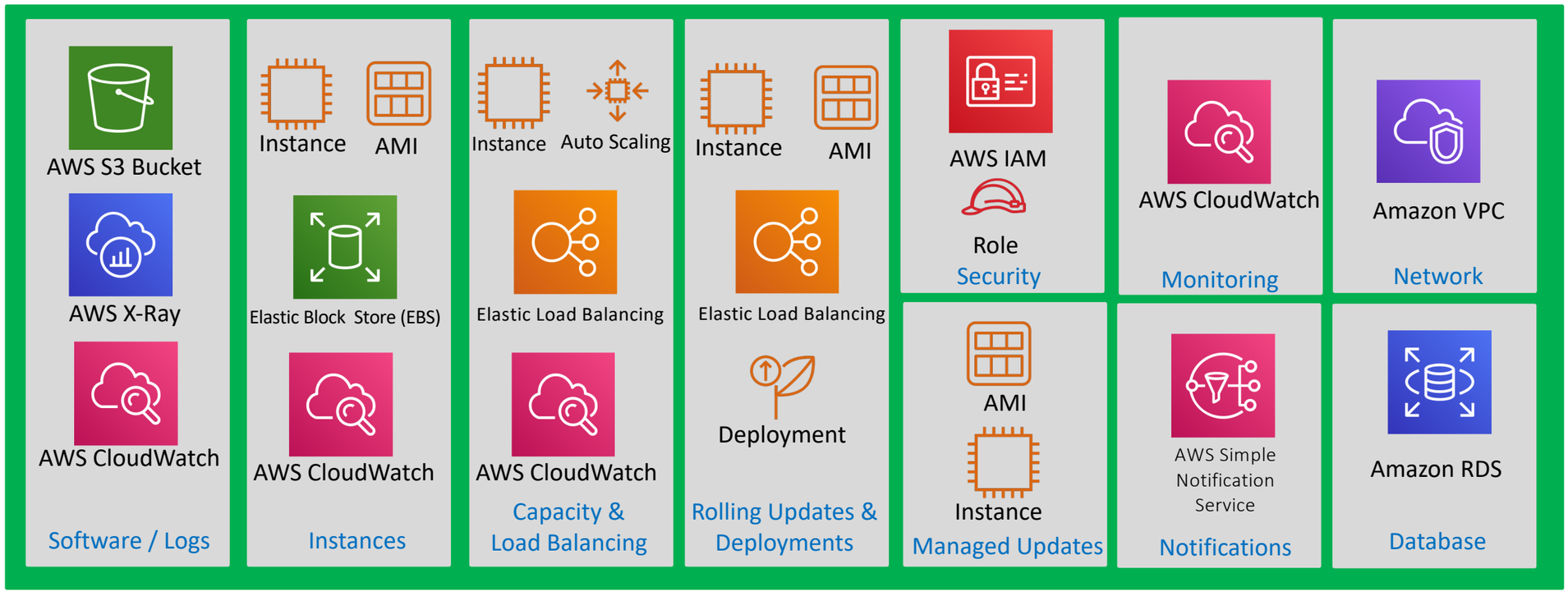
Dev | QA | Staging | Prod

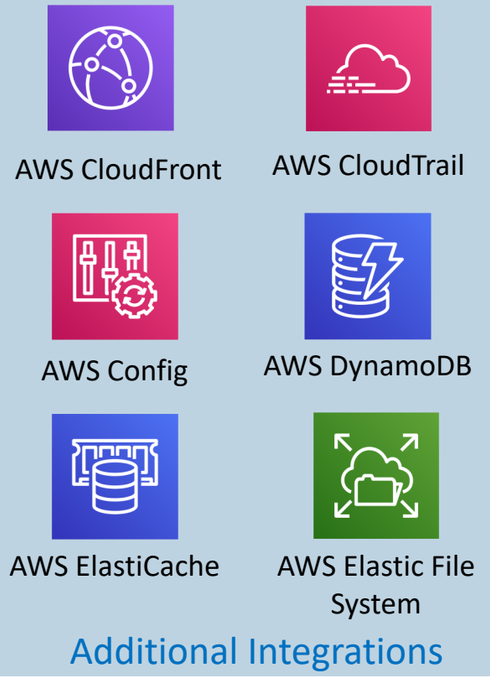
Environments

Application #2: Product Management

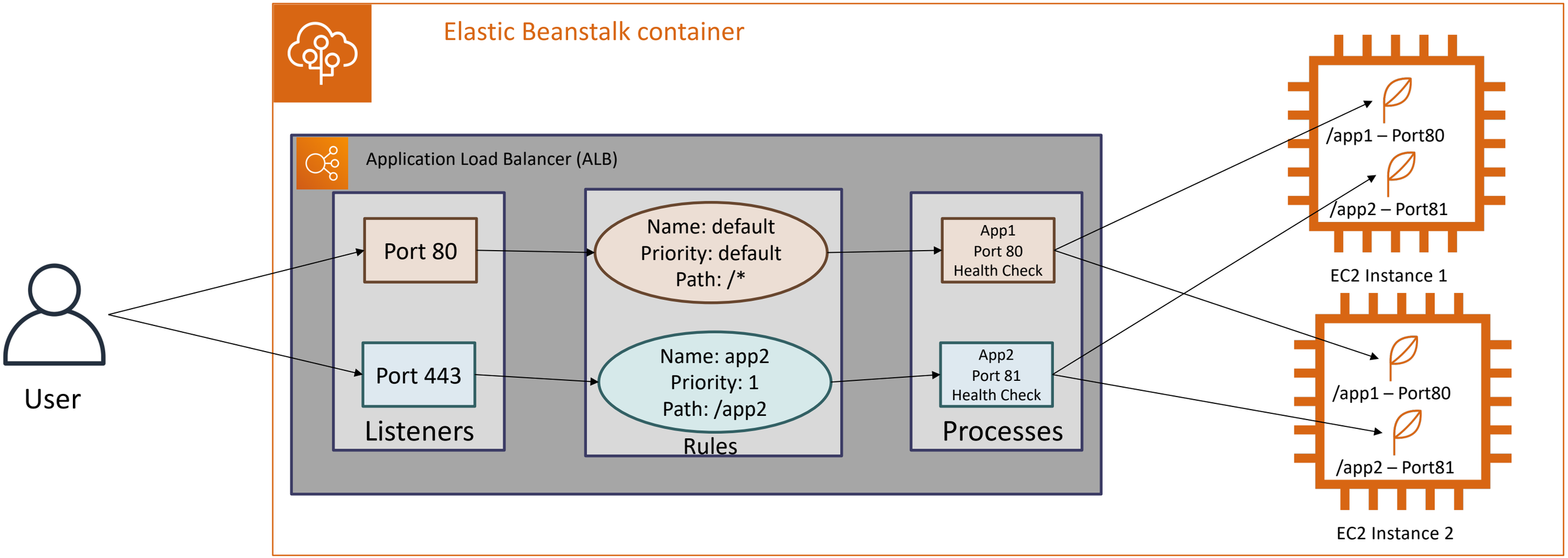
Dev | QA | Staging | Prod

Environments

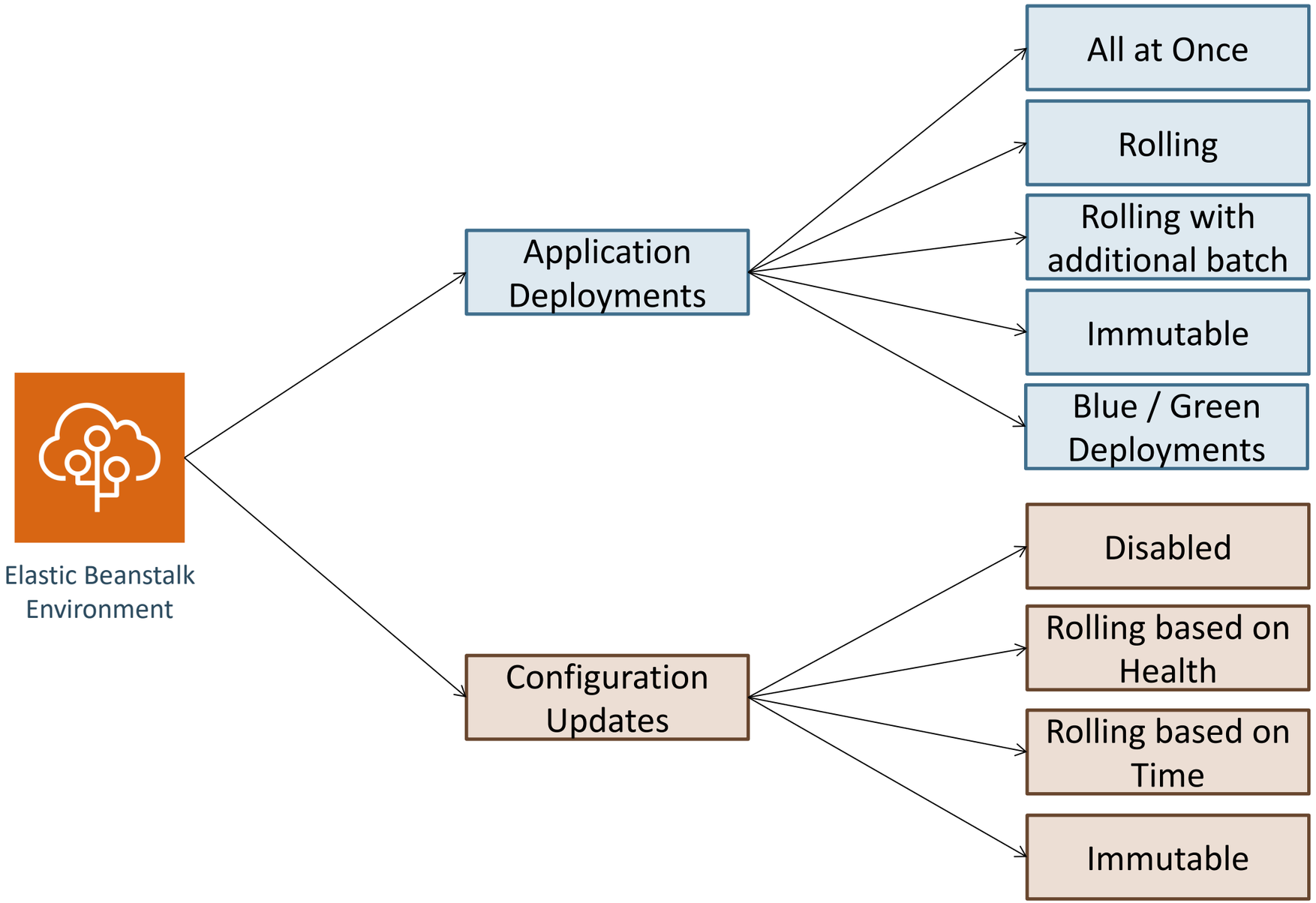




Application Load Balancer



Rolling Updates & Deployments



Which is the best Application Deployment Option?

Elastic Beanstalk Environment

Application Deployments

- **All at Once**
  - Service disruption during deployments
- **Rolling**
- **Rolling with additional batch**
  - In case of deployment failure, manual intervention is required
  - It ends up as partially completed deployment
- **Immutable**
  - Immutable deployments can prevent issues caused by partially completed rolling deployments.
  - If the new instances don't pass health checks, Elastic Beanstalk terminates them, leaving the original instances untouched.
- **Blue / Green Deployments**
  - This is useful for smaller applications whose other aws services integration scope is small.

Rolling Updates – Configuration Updates

Elastic Beanstalk Environment

Configuration Types

Replaces EC2 Instances

- Keypair change
- EC2 Instance Type change from t2.micro to something else
- Apply Managed Updates

No Impact to EC2 Instances

- Load Balancer Listener changes
- Load Balancer Health Status URL changes
- Environment Properties addition
- Add Notifications

Rolling Updates - Configurations

Elastic Beanstalk Environment

Configuration Updates (Rolling Update Type)

Disabled

- Updates applied all at once
- Instance replacement will be triggered
- Service Disruption

Rolling based on Health

- Updates happen in batches
- Minimum number of instances will be in service all times
- Moves to next batch once the current bacth health check passes

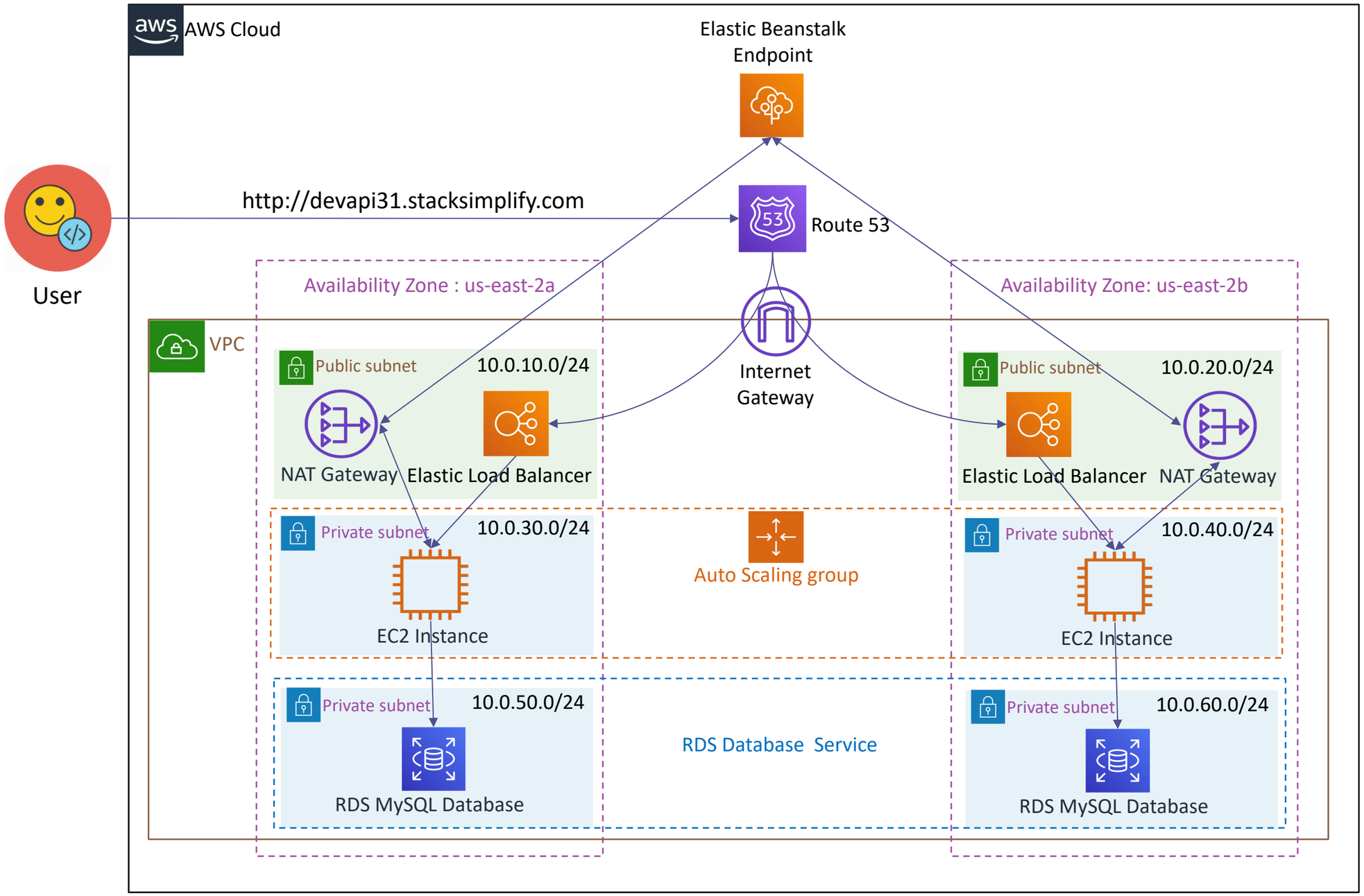
Rolling based on Time

- Updates happen in batches
- Minimum number of instances will be in service all times
- Moves to next batch once the Pause Time reaches

Immutable

- Same as Immutable Application Deployments

VPC Design

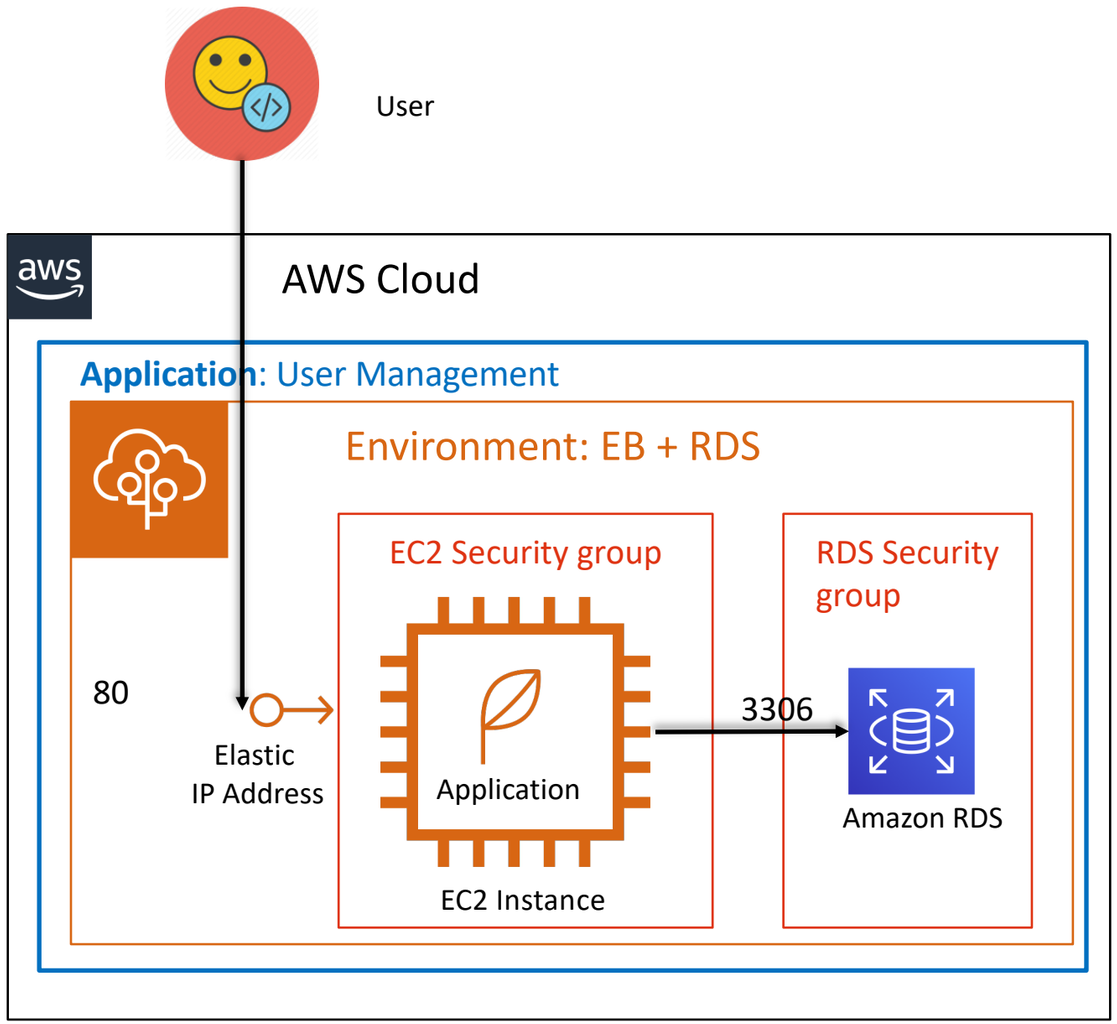


Public Routes	
Destination	**Target**
10.0.0.0/16	local
0.0.0.0/0	IGW

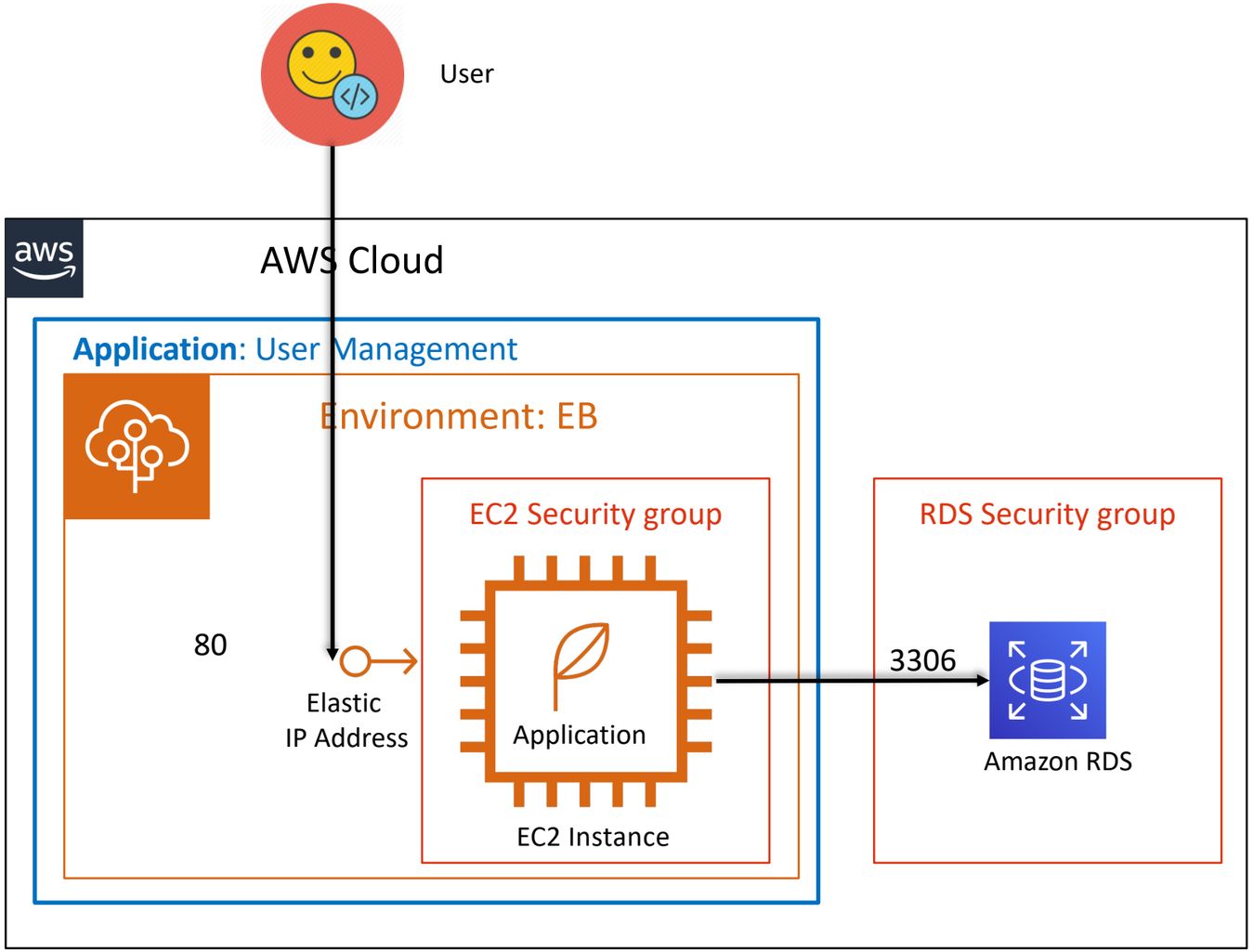
Private Instance 2a Routes	
Destination	**Target**
10.0.0.0/16	local
0.0.0.0/0	NAT-2a-GW

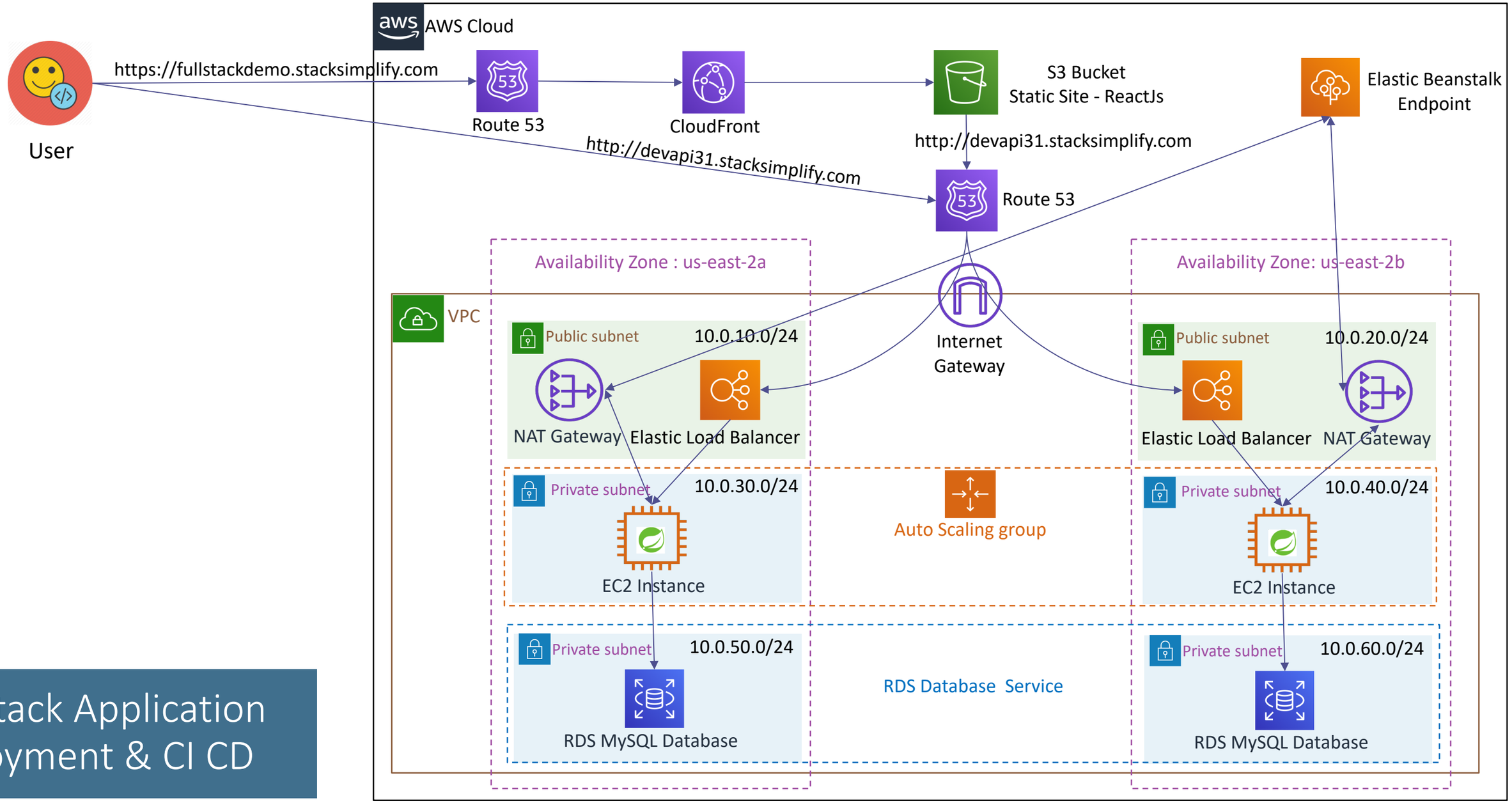
Private Instance 2b Routes	
Destination	**Target**
10.0.0.0/16	local
0.0.0.0/0	NAT-2b-GW

Option-1: RDS Database part of Elastic Beanstalk



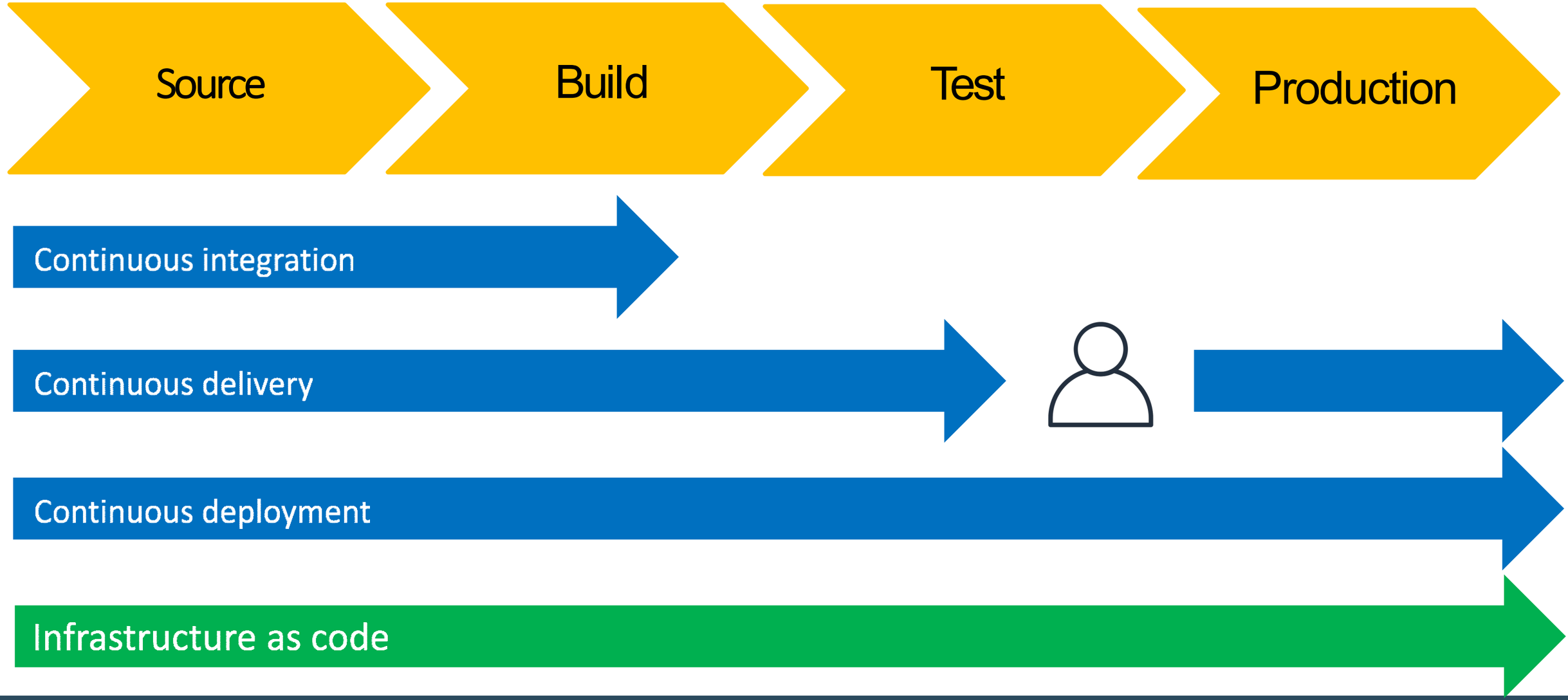
Option-2: RDS Database external to Elastic Beanstalk



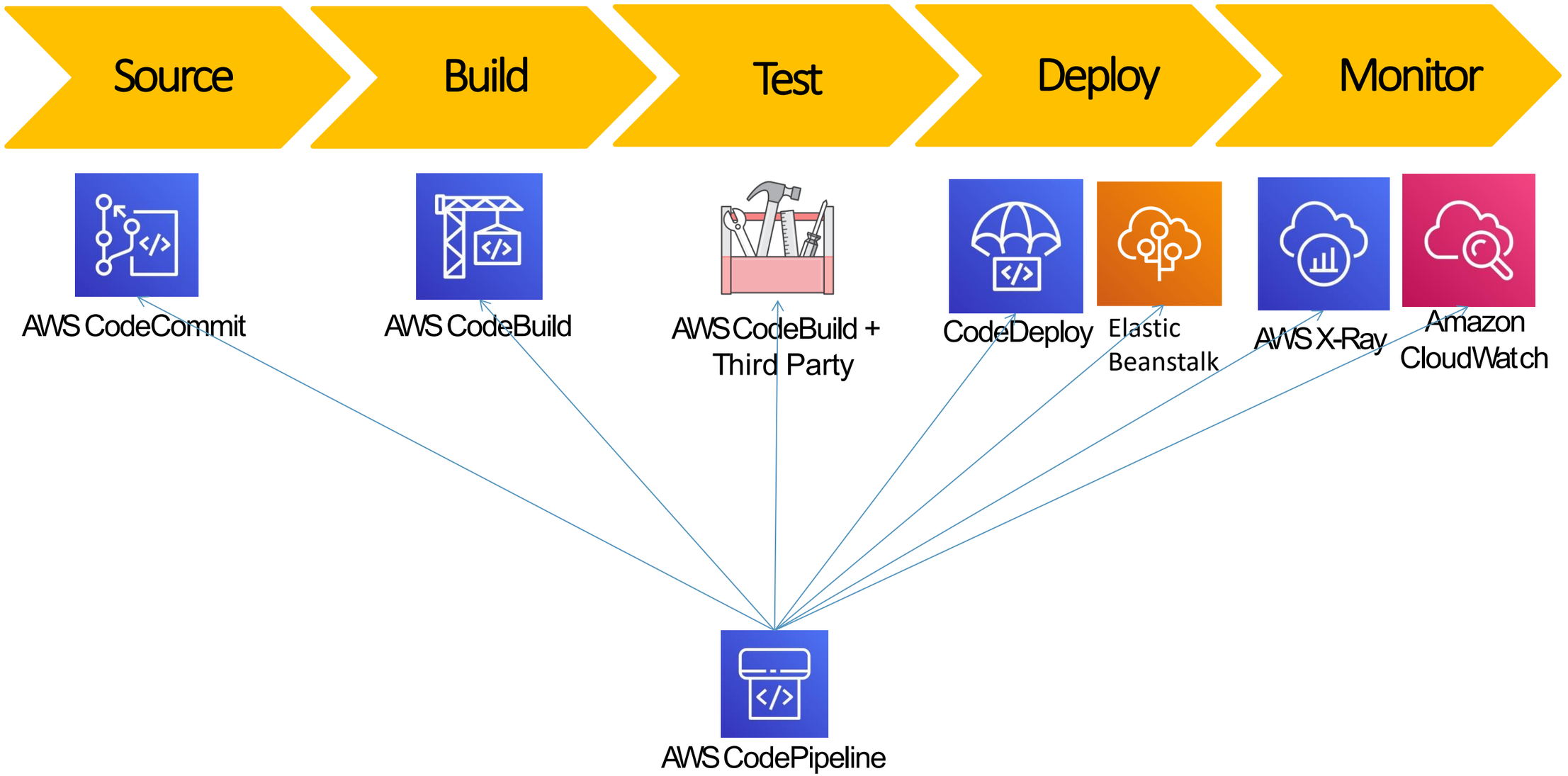


Full Stack Application Deployment & CI CD

Stages in Release Process



AWS Developer Tools or Code Services



AWS Developer Tools or Code Services

AWS CodeCommit

AWS CodeBuild

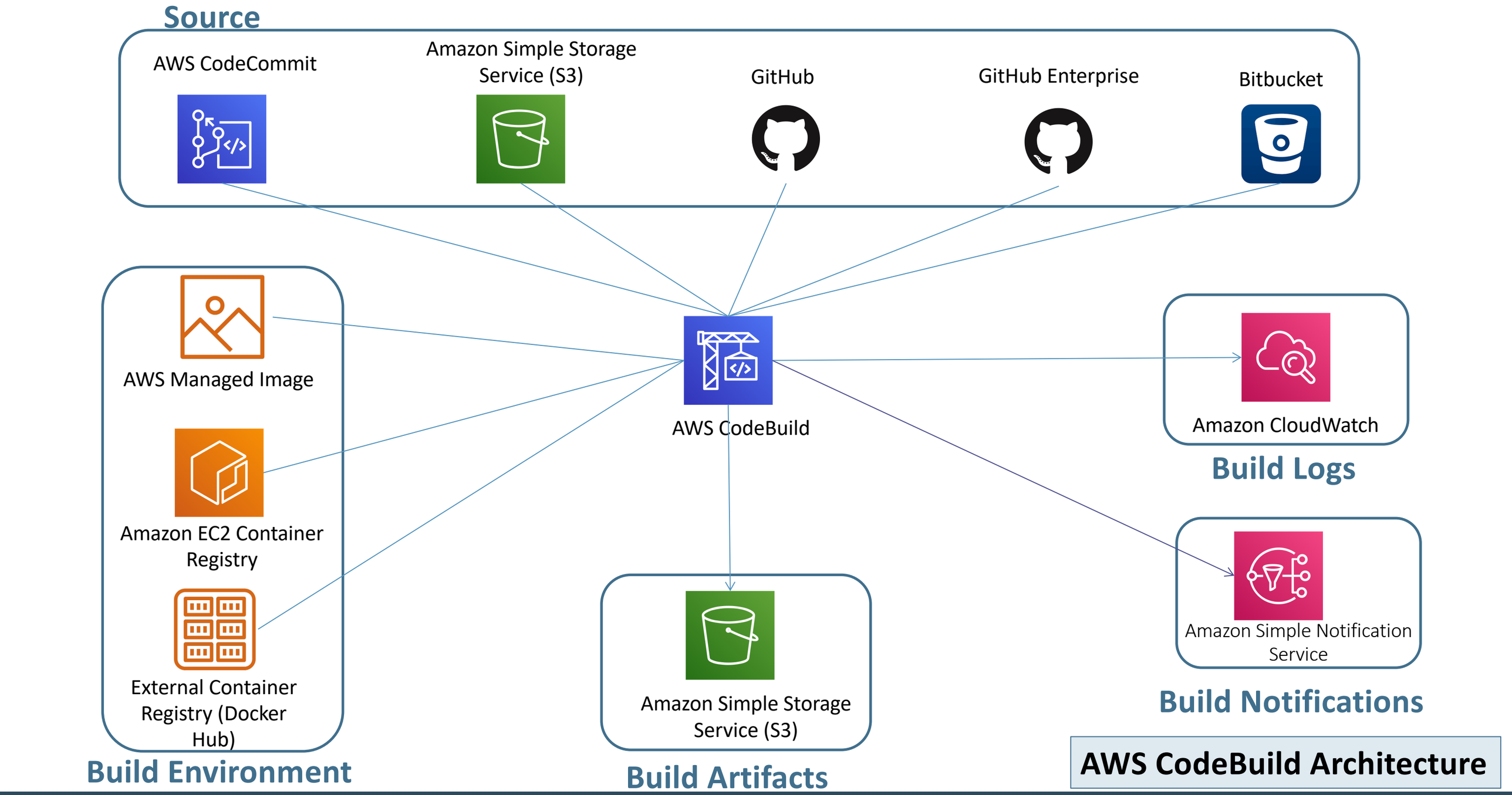
AWS CodeBuild + Third Party

Elastic Beanstalk

AWS X-Ray

Amazon CloudWatch

AWS CodePipeline


Source
AWS CodeCommit
Amazon Simple Storage Service (S3)
GitHub
GitHub Enterprise
Bitbucket
AWS Managed Image
Amazon EC2 Container Registry
External Container Registry (Docker Hub)
Build Environment
AWS CodeBuild
Amazon CloudWatch
Build Logs
Amazon Simple Storage Service (S3)
Build Artifacts
Amazon Simple Notification Service
Build Notifications
AWS CodeBuild Architecture

AWS CodePipeline Architecture

AWS CodePipeline

AWS CodeCommit

Amazon EC2 Container Registry

Simple Storage Service (S3)

GitHub

Source

AWS CodeBuild

Jenkins

Build

AWS CloudFormation

AWS CodeDeploy

AWS Elastic Beanstalk

AWS Service Catalog

Amazon Elastic Container Service

Amazon Elastic Container Service (Blue/Green)

Simple Storage Service (S3)

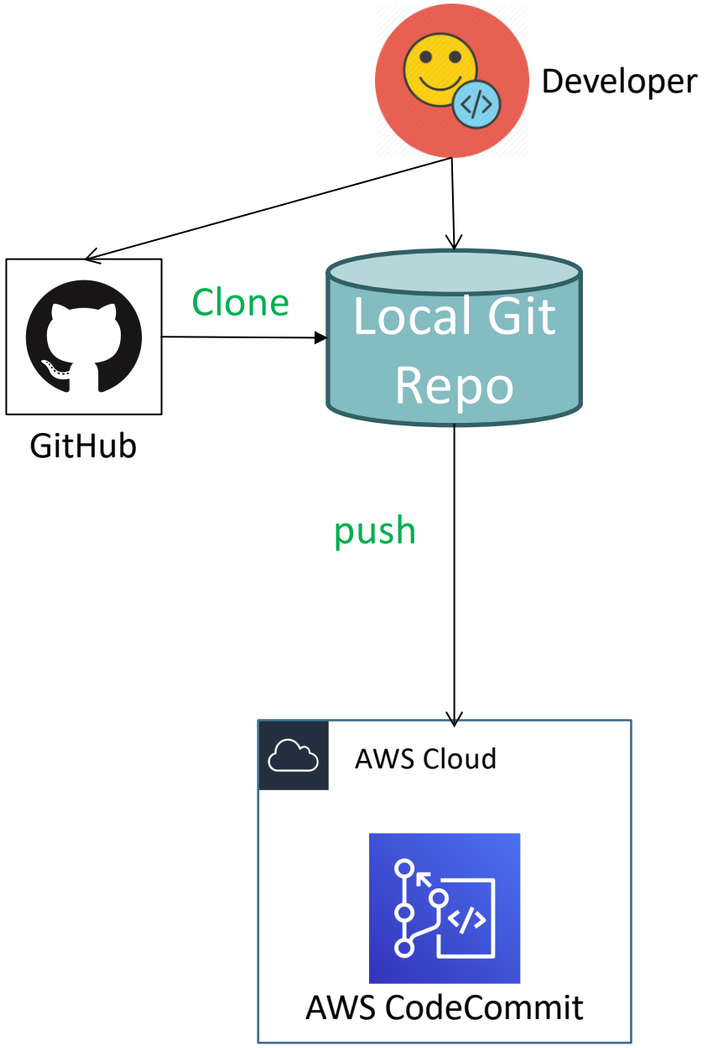
Deploy

Amazon CloudWatch

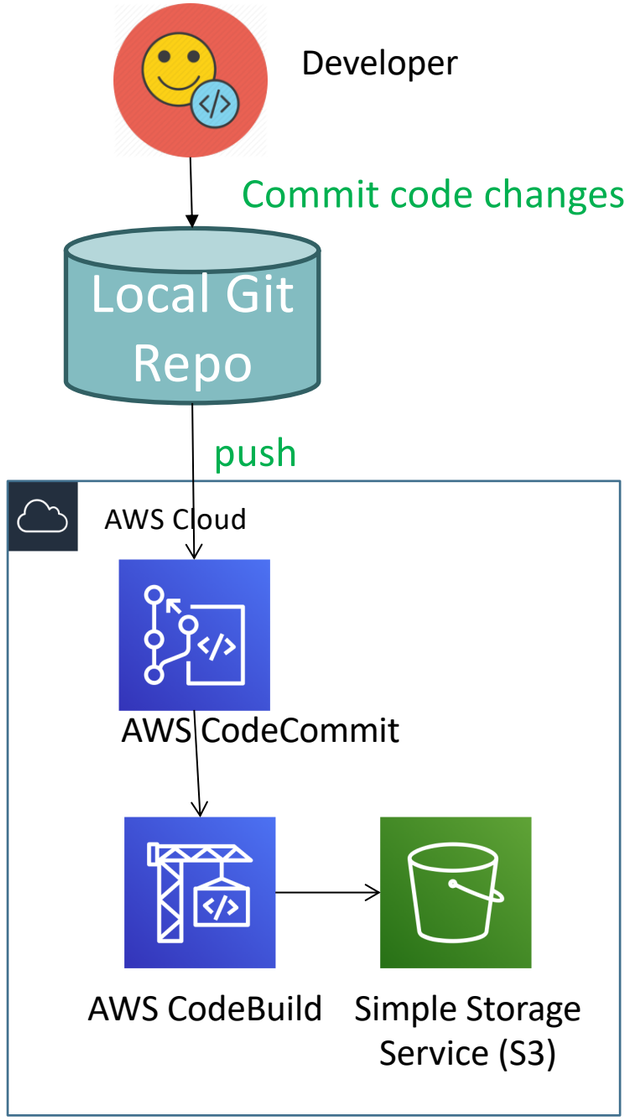
GitHub Webhooks

Monitor Source Changes

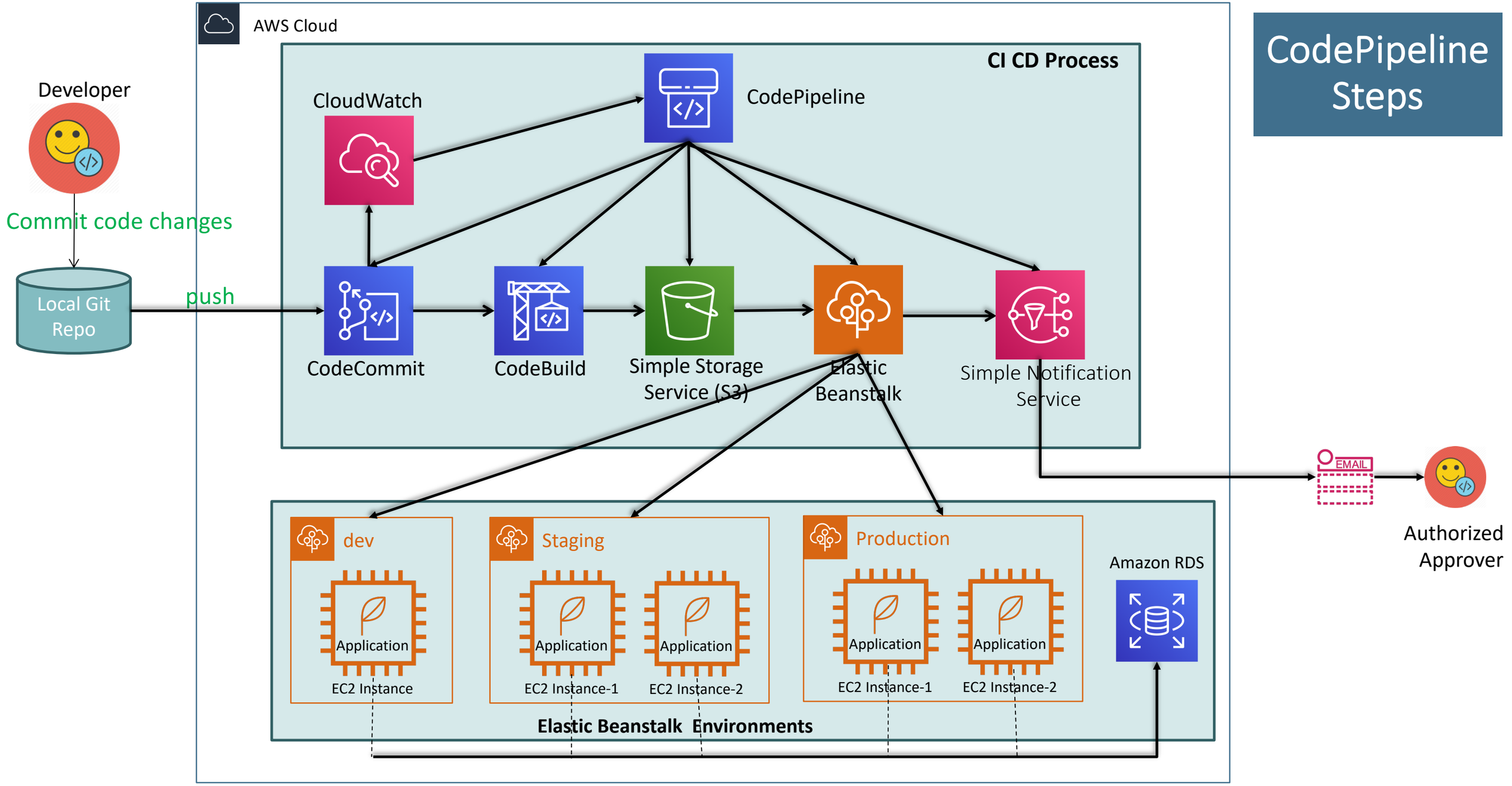
CodeCommit - Steps



CodeBuild - Steps



CodePipeline Steps



EB CLI Commands

- eb init
- eb status
- eb events
- eb health
- eb open
- eb list
- eb list –all
- eb terminate
- eb abort

- eb clone
- eb swap
- eb appversion
- eb logs
- eb scale
- eb deploy
- eb deploy –staged
- eb codesource codecommit

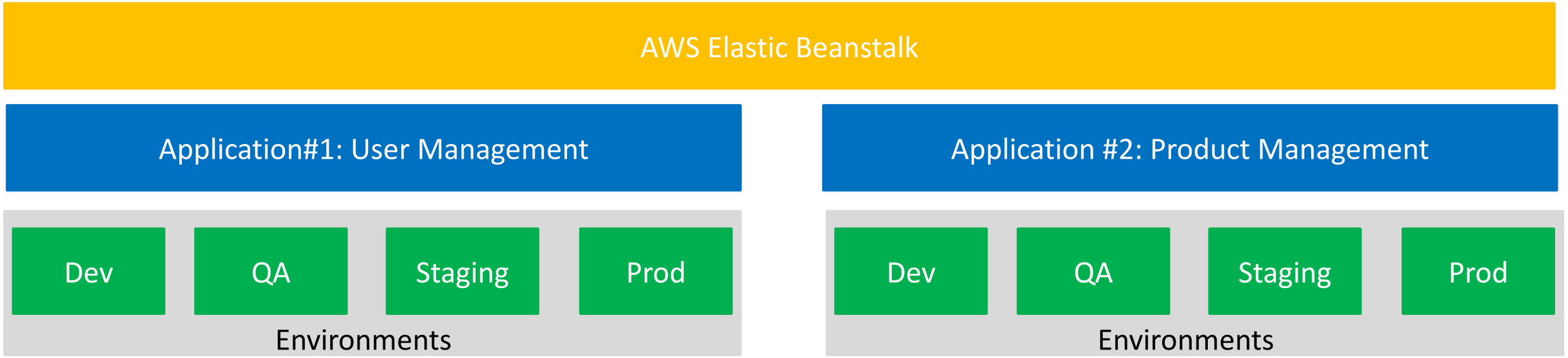
Elastic Beanstalk – Custom Platforms

Elastic Beanstalk
First Steps

Elastic Beanstalk

- We can quickly deploy and manage applications in the AWS Cloud without having to learn more about the infrastructure that runs those applications.
- In simple terms, we can consider learning Elastic Beanstalk as starting steps for AWS Cloud.
- Elastic Beanstalk reduces management complexity without restricting choice or control.
- We simply upload our application, and Elastic Beanstalk automatically handles the details of capacity provisioning, load balancing, scaling, and application health monitoring.

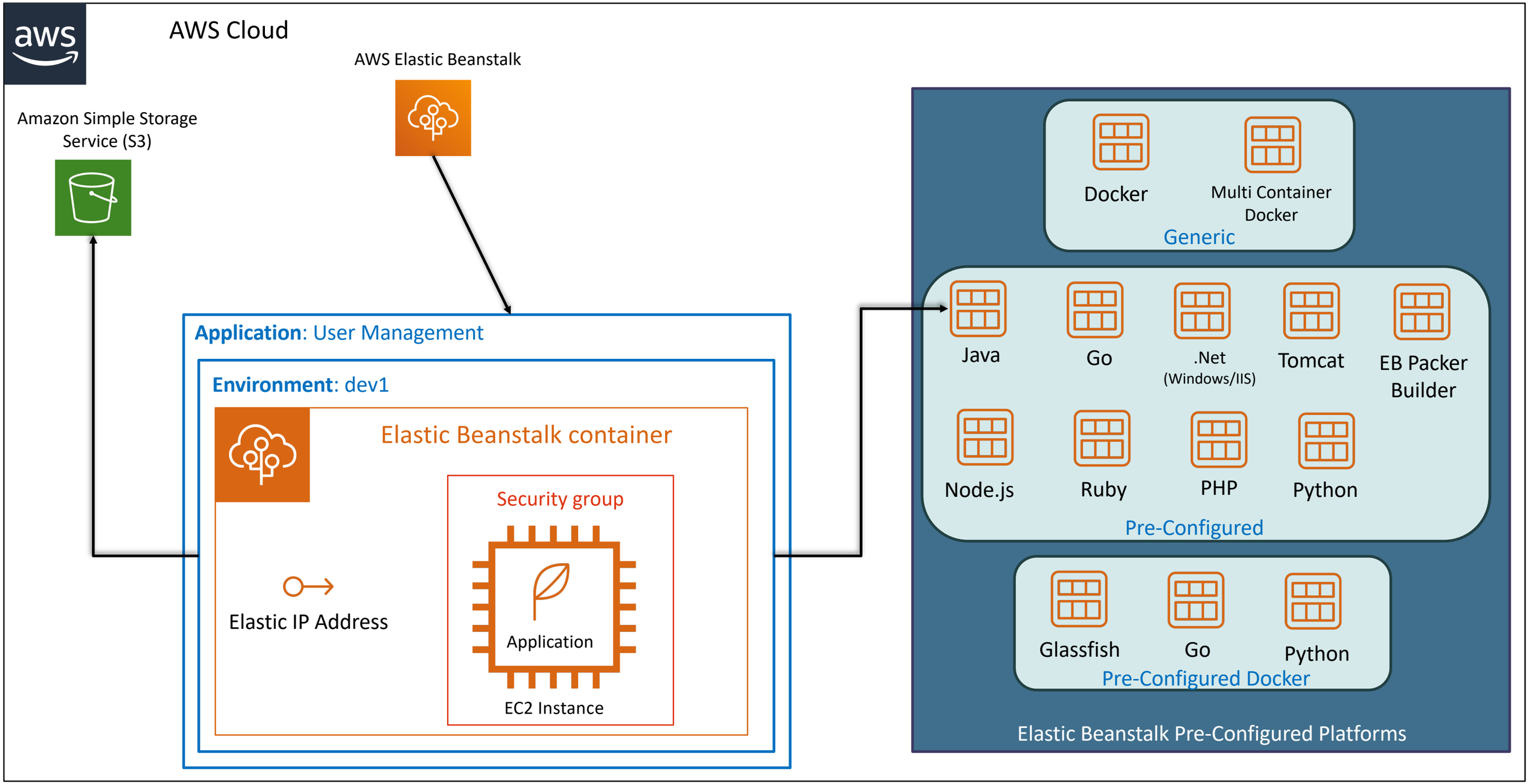
Elastic Beanstalk – First Steps



Elastic Beanstalk – First Steps

- Step-1: Create Application
- Step-2: Create Environment
  - Environment Type: Webserver
  - Preconfigured Platform: Java
  - Application Code: Sample Application
  - Test
- Step-3:Deploy a spring boot Rest API application
  - Jar file name: eb-usermgmt-h2.jar
  - Understand about
    - Application Versions
    - Version Life Cycle
- Step-4: Test the REST API's using postman
- Step-5: Associate Keypair to login to ec2 instance and understand the behavior
  - Existing EC2 instance gets terminated and new instance provisions immediately.

What happened in the background when a new environment is created using Elastic Beanstalk?


AWS Cloud
AWS Elastic Beanstalk
Amazon Simple Storage Service (S3)
Application: User Management
Environment: dev1
Elastic Beanstalk container
Security group
Elastic IP Address
Application
EC2 Instance
Docker
Multi Container Docker
Generic
Java
Go
.Net (Windows/IIS)
Tomcat
EB Packer Builder
Node.js
Ruby
PHP
Python
Pre-Configured
Glassfish
Go
Python
Pre-Configured Docker
Elastic Beanstalk Pre-Configured Platforms

What happened in the back ground?

- Step-1: Verify S3 for uploaded artifacts (jar file)
- Step-2: Verify EC2 Instance
  - Roles
  - Security Groups (Port 22)
  - Elastic IP (Per region default limit of elastic IP is 5)
- Step-3: Login to EC2 Instance
  - Verify /opt/elasticbeanstalk (appsource)
  - Verify /var/app/current
  - Verify logs in ec2 instance /var/log
    - eb-*.log
    - web*.log
    - cloud-init.log
  - Verify logs via EB Console

Elastic Beanstalk Environment Features

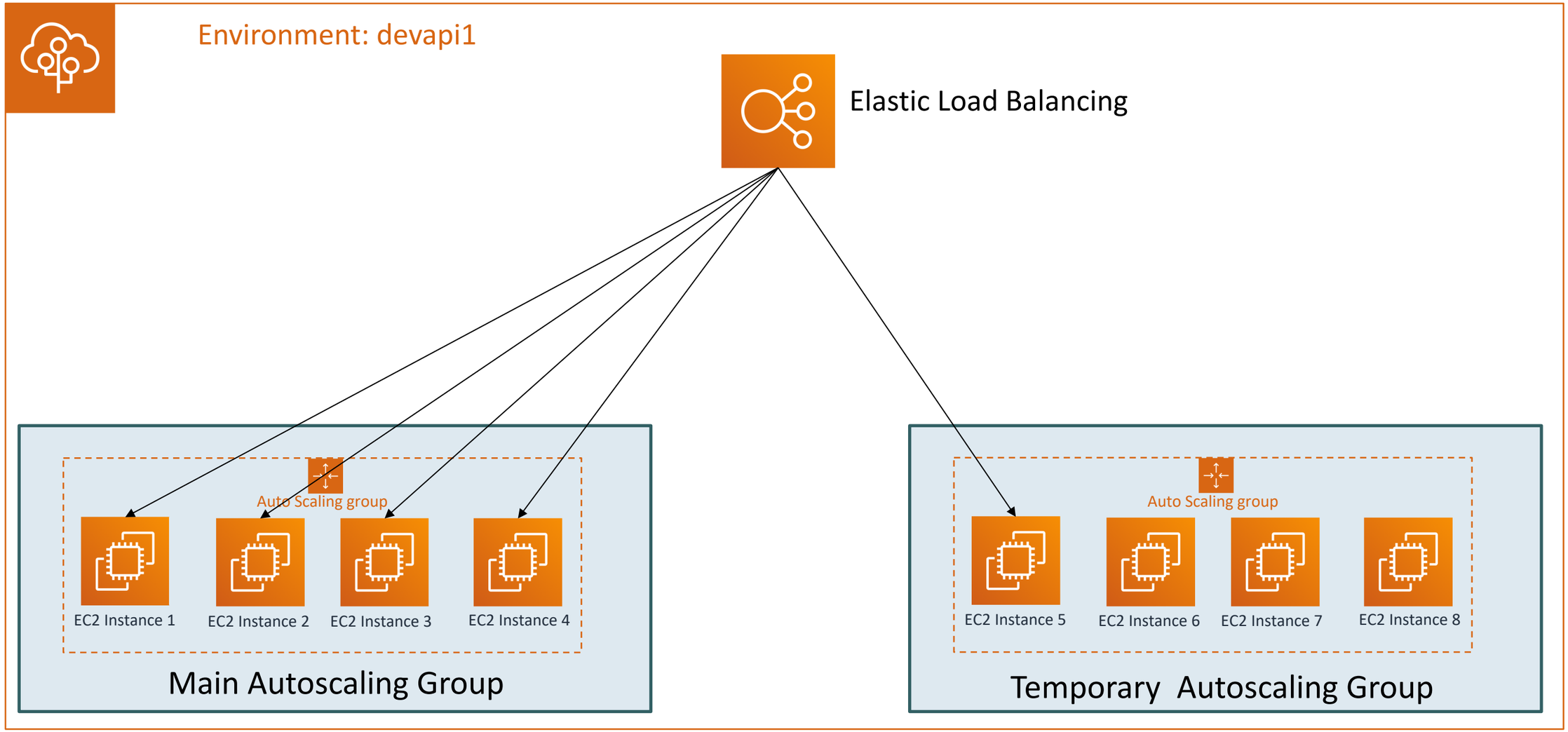
EB Environment Features

AWS Elastic Beanstalk Environment

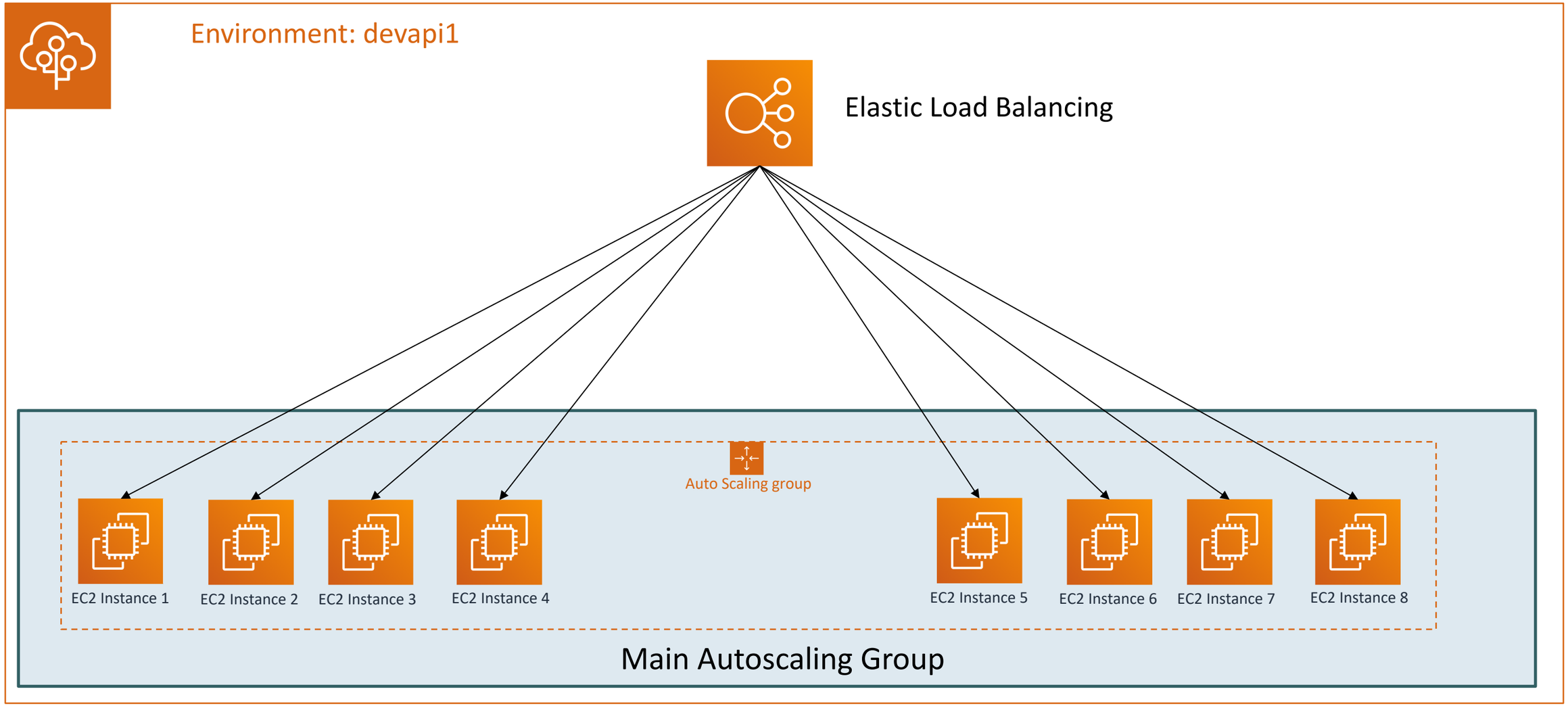
- Dashboard
- Configuration
- Logs
- Health
- Monitoring
- Alarms
- Managed Updates
- Events
- Tags
- Actions

- Save Configuration
- Load Configuration
- Swap Environment URLs
- Clone Environment
- Clone with Latest Platform
- Abort Current Operation
- Restart App Server(s)
- Rebuild Environment
- Terminate Environment
- Restore Terminated Environment (From Applications screen)

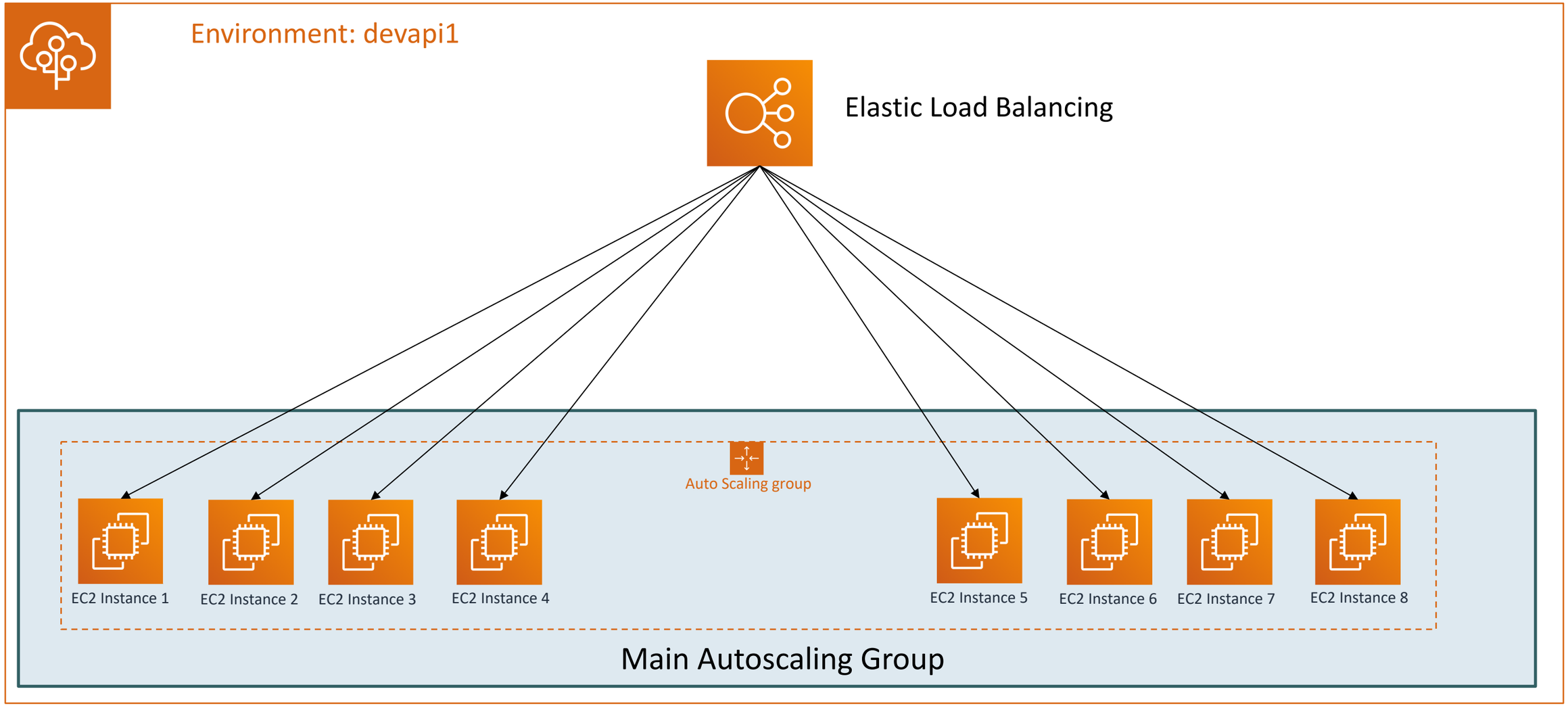
Managed Updates – Immutable Environment Updates



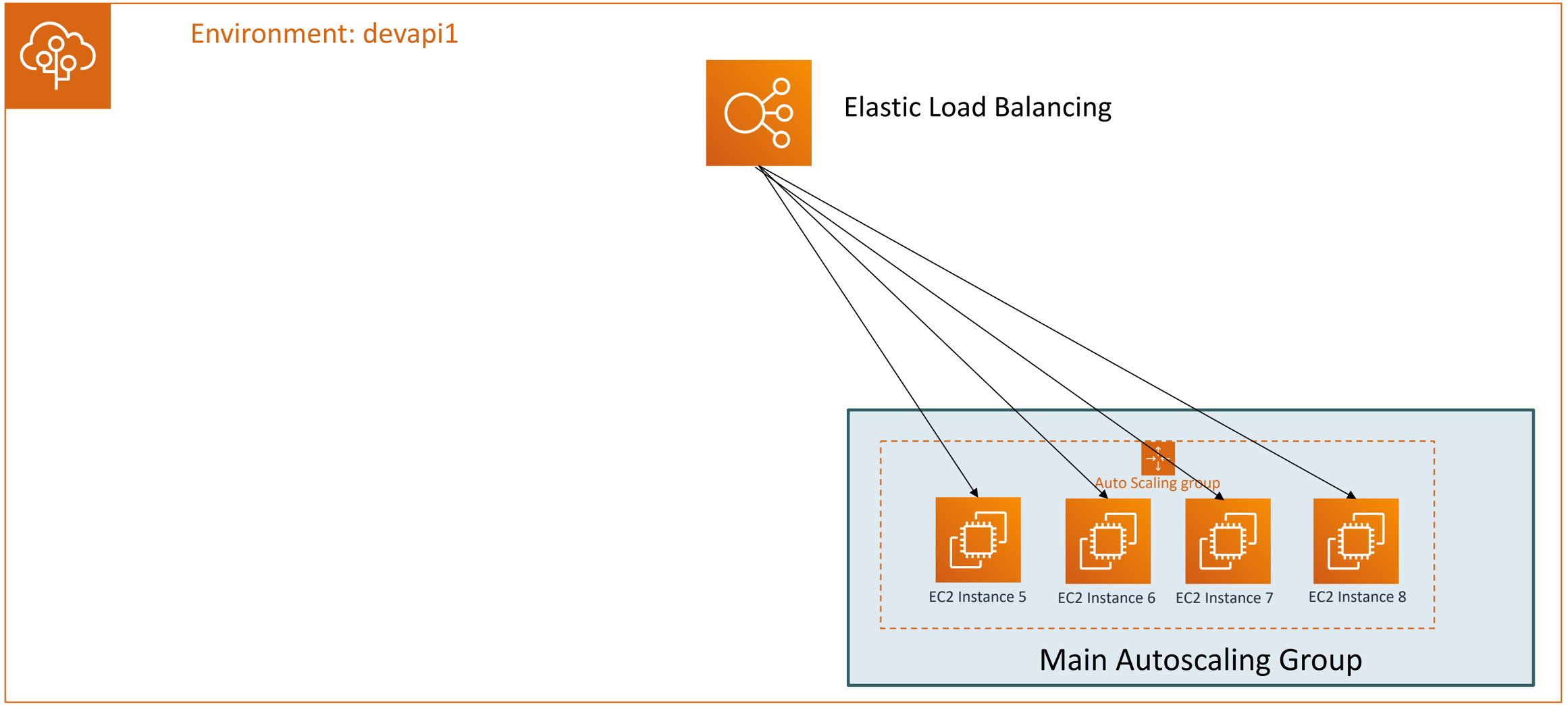
Managed Updates – Immutable Environment Updates



Managed Updates – Immutable Environment Updates

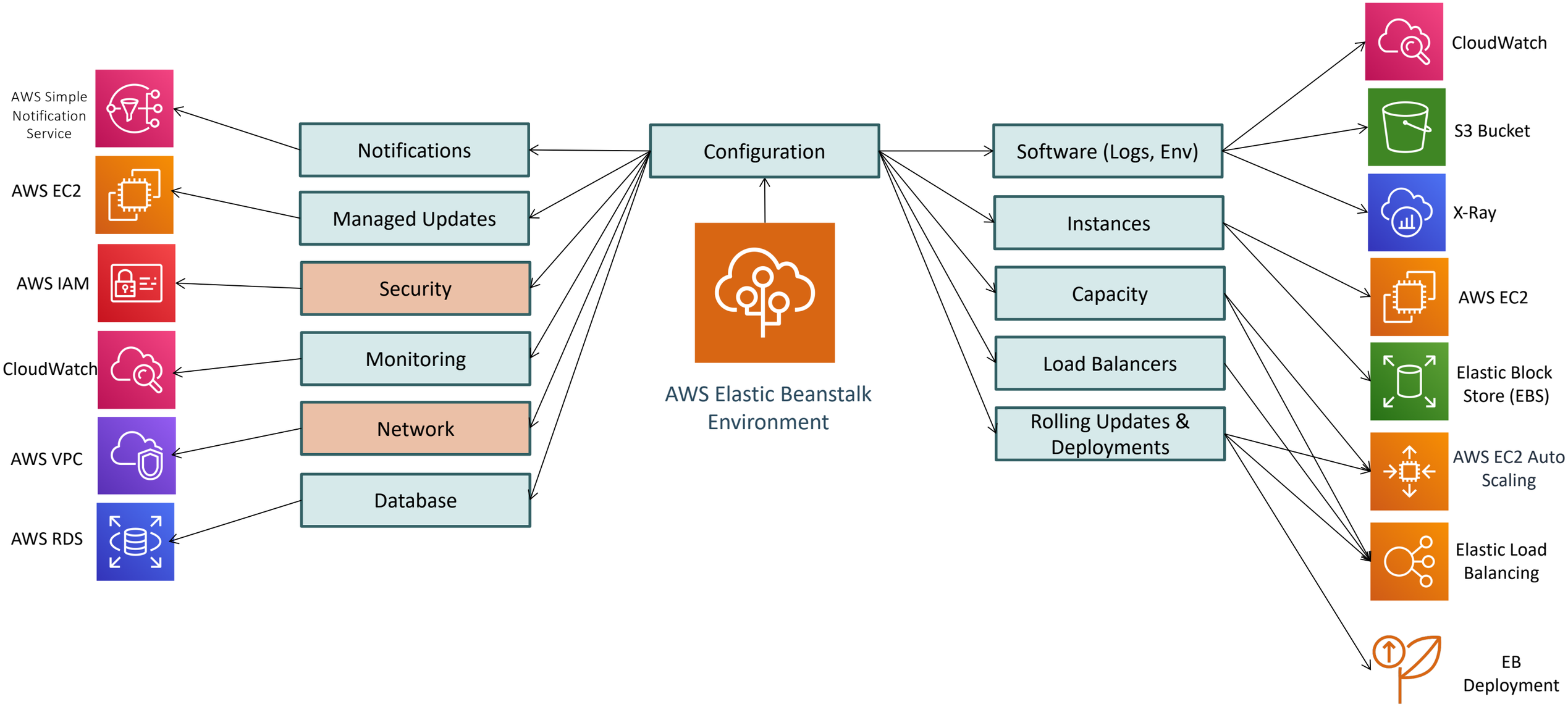


Managed Updates – Immutable Environment Updates



Elastic Beanstalk Environment Configuration

Elastic Beanstalk Environment - Configuration



AWS Elastic Beanstalk

Application#1: User Management

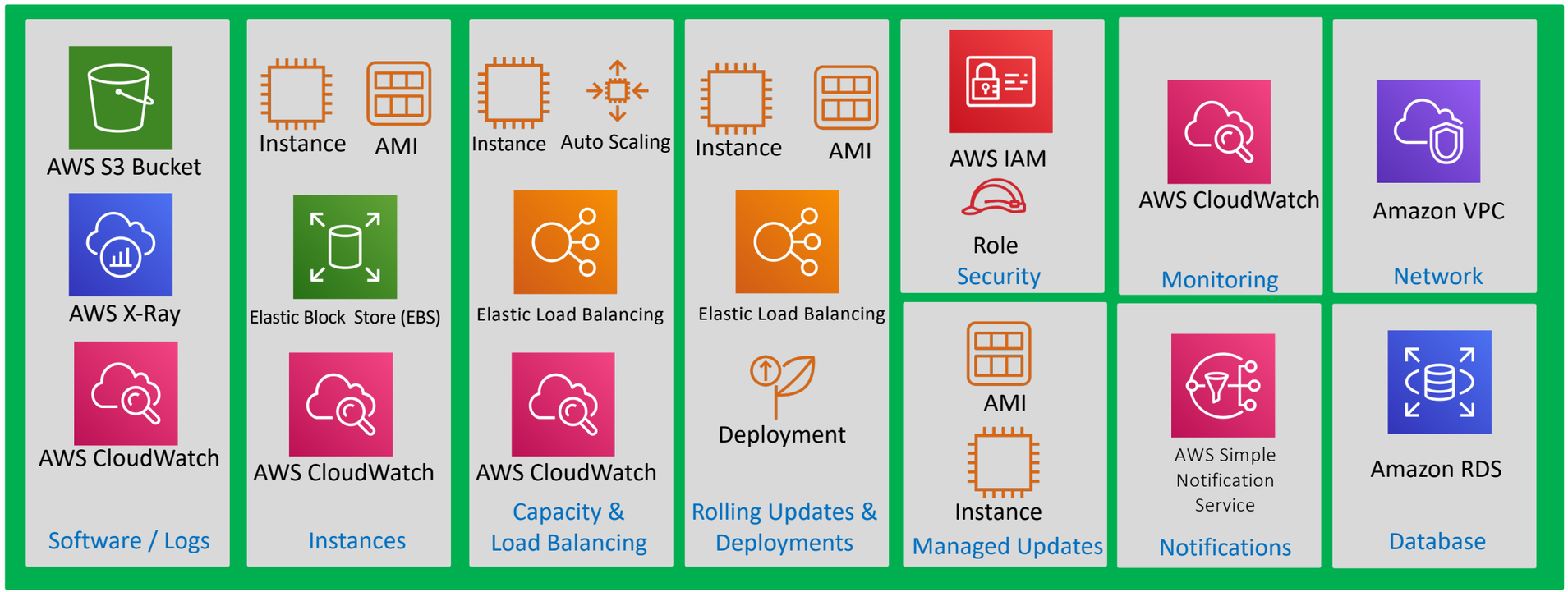
Dev | QA | Staging | Prod

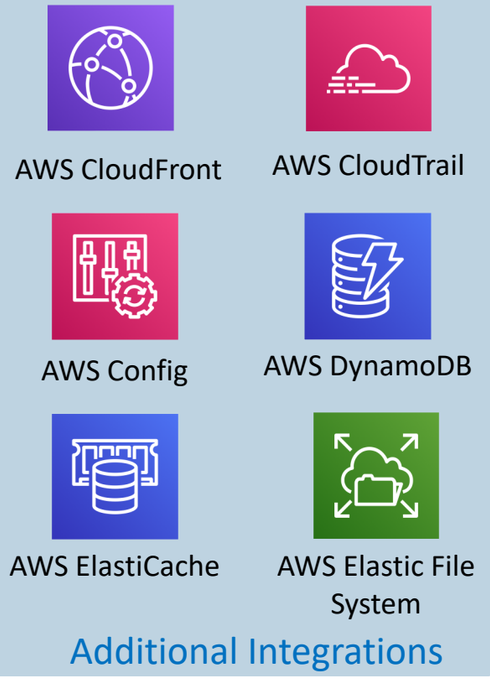
Environments

Application #2: Product Management

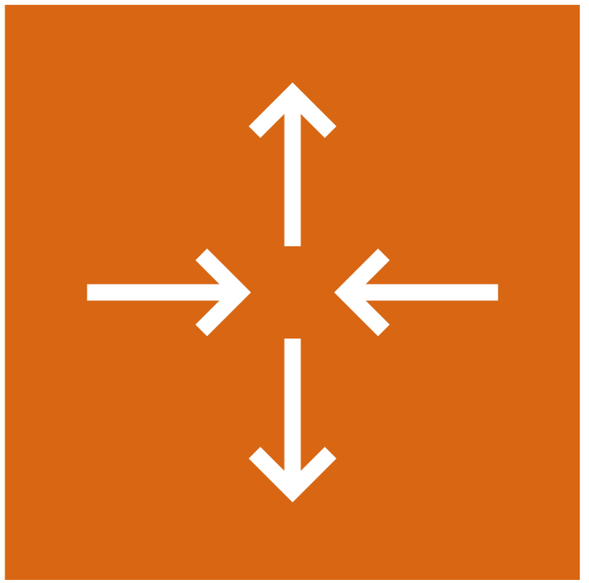
Dev | QA | Staging | Prod

Environments





Elastic Beanstalk Environment Configuration



Capacity - Autoscaling Groups – Scheduled Actions

Capacity - Scheduled Autoscaling Actions

- We can configure our environment with a recurring action to scale up each day in the morning, and scale down at night when traffic is low.
- In short, we can heavily save costs using these scheduled actions if we use them in combination with our business knowledge (low traffic times, peak traffic times) on applications deployed in this environment.
- We can schedule up to 120 scheduled actions per environment
- EB also retains 150 expired scheduled actions which we can reuse by updating their actions.
- Two types of scheduled actions
  - One-Time
  - Recurring (uses cron)

Capacity - Scheduled Autoscaling Actions

- One-Time Action
  - Name: IncreaseCapacity
  - Min Servers: 7
  - Max Servers: 10
  - Occurrence: One-Time
  - Start Time: UTC timezone

- **Recurring Action**
  - Name: LowTrafficPeriod
  - Min Servers: 2
  - Max Servers: 3
  - Occurrence: Recurring
  - Recurrence: 00 20 * * *
  - Start Time: Optional
  - End Time: Optional

- **Start time for Recurrent Actions**
  - For recurrent actions, a start time is optional.
  - Specify it to choose when to activate the action.
  - If not specified, the action is activated immediately, and recurs according to the **Recurrence** expression.

- **Recurring Action**
  - Name: PeakTrafficPeriod
  - Min Servers: 4
  - Max Servers: 6
  - Occurrence: Recurring
  - Recurrence: 00 8 * * *
  - Start Time: Optional
  - End Time: Optional

Elastic Beanstalk Environment Configuration

Load Balancers

Classic Load Balancer (CLB)

- Supports HTTP, HTTPS & TCP
- Listeners
- Sessions
- Cross Zone Load Balancing
- Connection Draining
- Health Check
- All traffic to a listener is routed to a single port on the backend instances (Major drawback when compared to ALB)
- Not cost effective

Application Load Balancer (ALB)

- Supports HTTP & HTTPS
- Listeners
- Processes
- Rules
- Health Check
- Sessions
- Access Log Capture and push to S3
- Can't have transport layer (layer 4) TCP or SSL/TLS listeners.
- **URI Routing:** Direct traffic for certain paths to a different port on webserver.
- Cost Effective – single load balancer can be used for multiple applications listening on different ports in EC2 Instances.

Network Load Balancer (NLB)

- Supports TCP only (layer4)
- Highly performant
- No layer7 HTTP or HTTPS
- Listeners
- Processes (No Rules)
- Health Check: Supports active and passive health checks.

Elastic Beanstalk Environment Configuration

Application Load Balancer (ALB)

Application Load Balancer (ALB)

- Load balancer is to distribute traffic among the instances in our environment.
- URI Routing (Core Feature of ALB): Application Load Balancer inspects traffic at the application network protocol layer to identify the request's path (/admin, /user) so that it can direct requests for different paths to different destinations. CLB lacks this feature.
- Default is to accept requests on Port 80 and distribute them to instances in our environment.
- We need to understand about 3 things to master ALB from Elastic Beanstalk perspective
  - Listeners
  - Processes
  - Rules

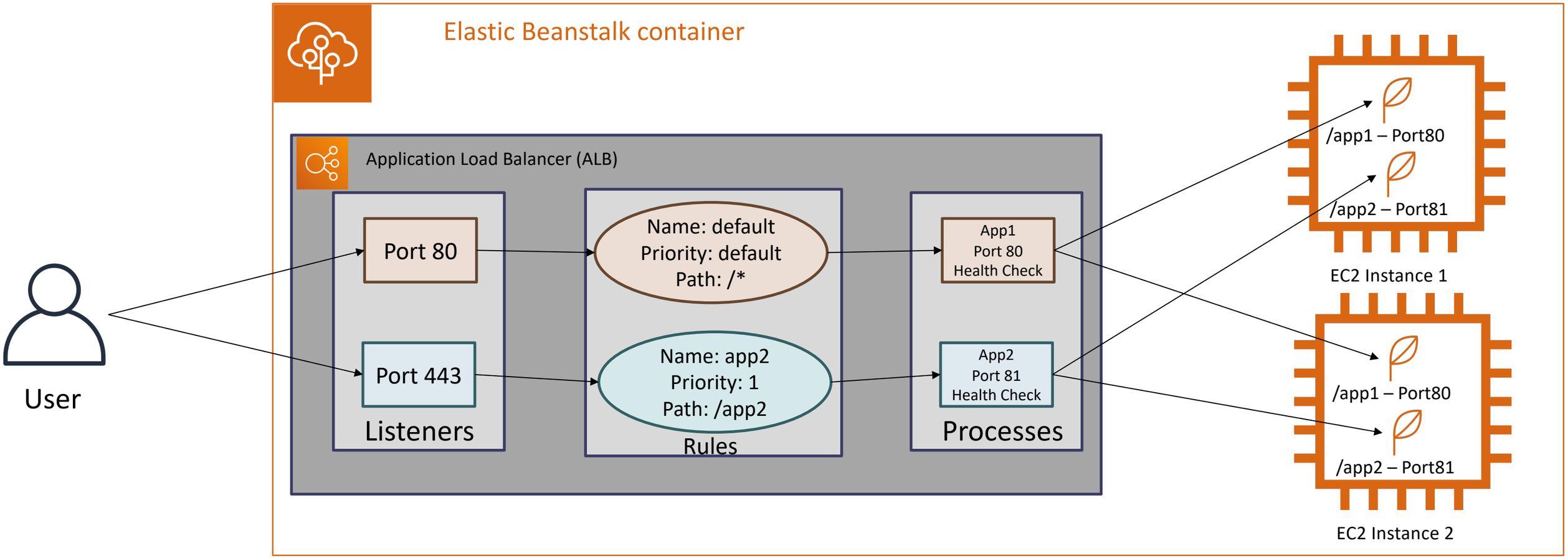
Application Load Balancer (ALB)

- Listener
  - Each listener routes incoming client traffic on a specified port using a specified protocol to one or more processes on our instances
  - In simple terms, we can call it as a load balancer port where client traffic is routed.
  - As soon as ALB created a default listener, a default process and a default rule gets created which routes incoming HTTP traffic on port 80 to a process named **default**, which listens to HTTP port 80.
- Processes
  - A process is a target for listeners to route traffic. In simple terms we can call it as a backend instances port configured on ALB as a process.
  - Health Check
    - To configure backend instances health check.
  - Sessions
    - Persistence / Stickiness configuration – It will let us control whether the load balancer routes requests for the same session to the same EC2 Instance. Primarily deals with the persistence named cookie persistence

Application Load Balancer (ALB)

- Rules
  - A rule maps requests that the listener receives on a specific path pattern to a target process.
  - Each listener can have multiple rules, routing requests on different paths to different processes on our instances.
  - Rules have numeric priorities that determine the precedence in which they are applied to incoming requests.
  - For each new listener we add, Elastic Beanstalk adds a default rule that routes all the listener's traffic to the default process
  - The default rule's precedence is the lowest; it's applied if no other rule for the same listener matches the incoming request.

Application Load Balancer



Application Load Balancer

- Step-1: Select Application Load Balancer when creating environment. Leave all settings to default
- Step-2: Identify the failures with default root context of application and fix the health status (/health/status) in default process.
- Step-3: Enable SSL on ALB (SSL terminated on ALB)
  - Create SSL certificate using Certificate Manager
  - Create Listener with port 443
  - Verify the default rule associated with port 443 listener and default process on port 80
  - Apply changes and Test

Application Load Balancer (ALB)

- Step-4: Store ALB Access Log files to S3
  - Create a S3 bucket
  - Associate the ELB policy to s3 bucket
  - In EB environment, enable the “Store logs”.
  - Test by accessing some API’s and Verify access logs in S3 bucket

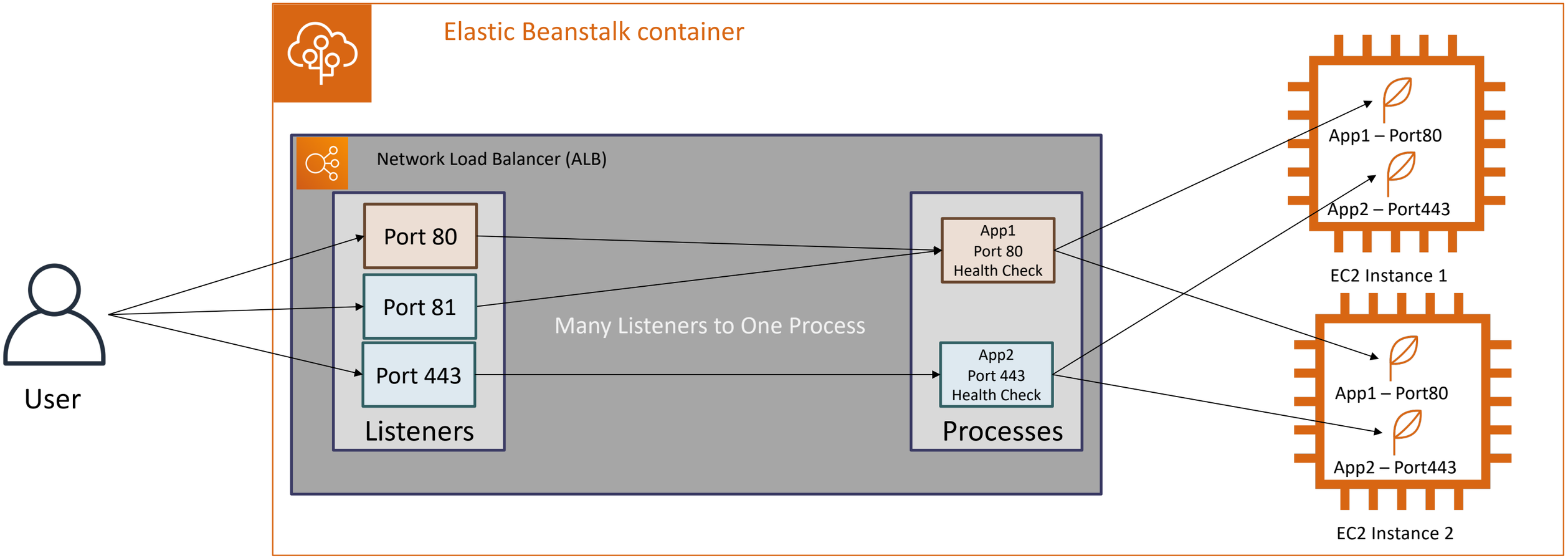
Elastic Beanstalk Environment Configuration

Network Load Balancer (NLB)

Network Load Balancer (NLB)

- Network load balancers only serves plain TCP Traffic
- The default listener accepts TCP requests on port 80 and distributes them to the instances in our environment
- Network Load Balancer supports active health checks. These checks are based on messages to the root (/) path.
- In addition, Network Load Balancer also supports passive health checks. It automatically detects faulty backend instances and routes traffic only to healthy instances.
- Doesn’t support HTTP or HTTPS
- Doesn’t support SSL termination on Load balancer. It can act as a plain TCP proxy for even SSL connections wherein SSL termination happens at application level.

Network Load Balancer (NLB)



Network Load Balancer - Demo

- Step-1: Create a new environment with network load balancer configuration
  - Discuss about NLB Listeners and Processes
  - Discuss about its ”Many Listeners to One Process” Association

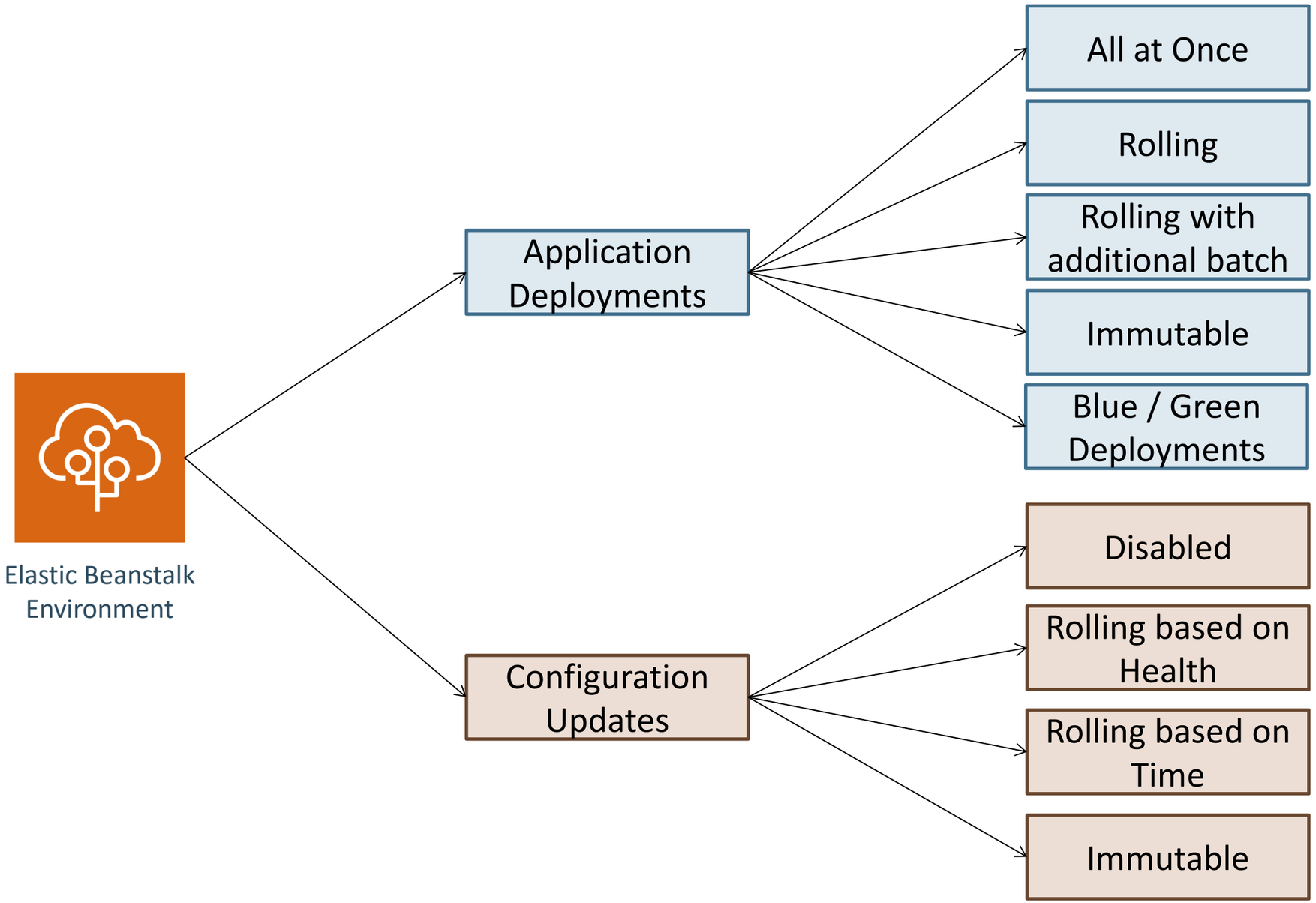
Important Point about Load Balancers

- New Environment: When we create the load balanced environment during environment creation we have the choice to select one load balancer among the three
  - Application Load Balancer
  - Network Load Balancer
  - Classic Load Balancer
- Converting existing Single Instance Environment: When we convert a single instance environment to load balanced environment by changing the configuration in Capacity Section, elastic beanstalk automatically by default creates the Classic Load Balancer.

Elastic Beanstalk Environment Configuration

Rolling Updates & Deployments

Rolling Updates & Deployments

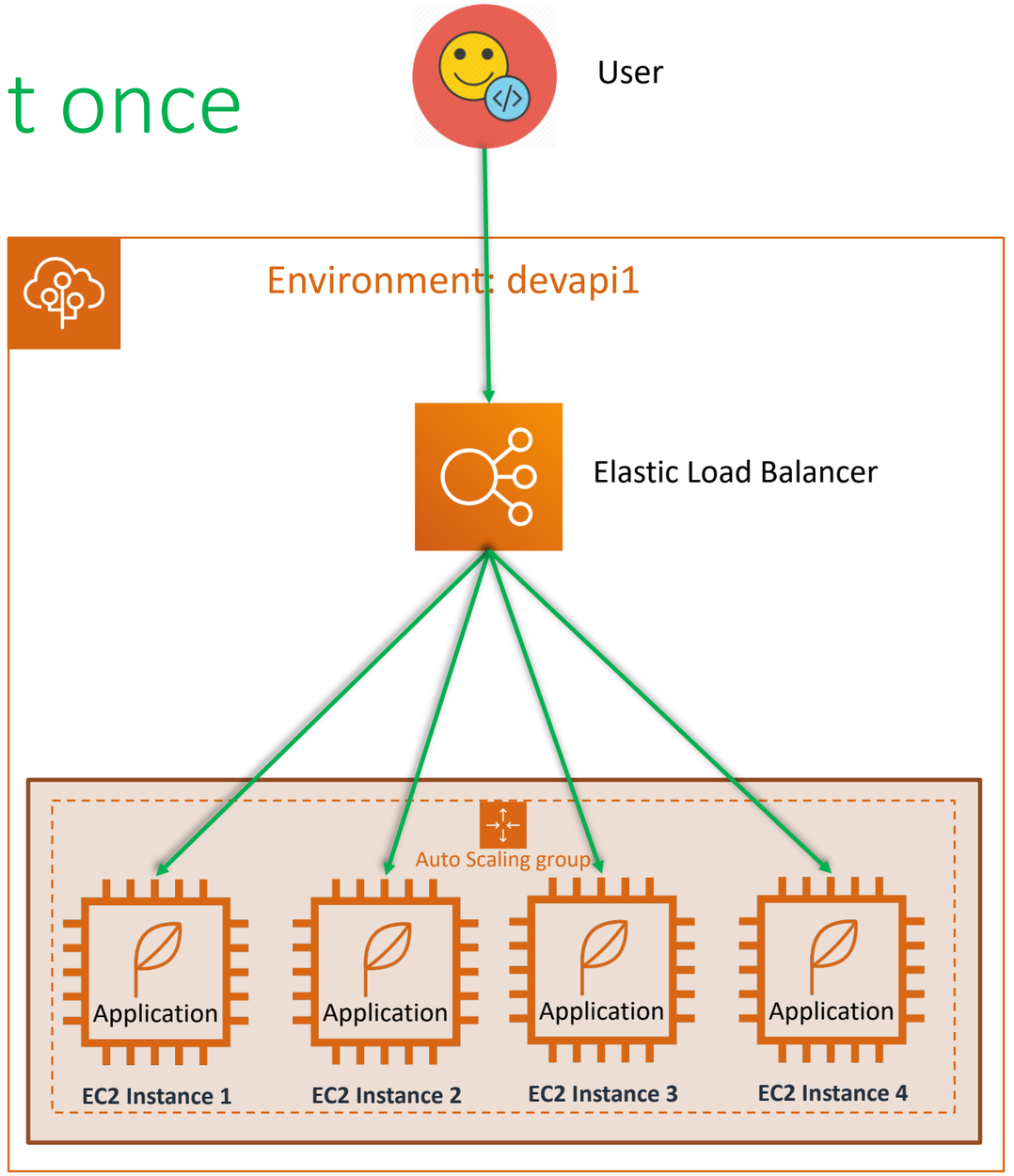


Create Environment – Rolling Deployments Enabled

- Step-1: Upload 3 versions of code for effective testing and understanding features.
  - Location: EB-Application-Jars/RollingUpdates-Deployments
    - eb-appdeployments-h2-v1.jar
    - eb-appdeployments-h2-v2.jar
    - eb-appdeployments-h2-v3.jar
- Step-2: Create Environment
  - Environment Tier: Webserver
  - Preconfigured Platform: Java
  - Application Code: Existing Version
  - Configuration Presets: High Availability
  - Rolling Updates & Deployments:
    - Application Deployments
      - Deployment Policy: Rolling
  - Important Note-1 & Tip: During environment creation if "All at Once" selected, Application Deployments section itself will not be displayed after environment creation. So if we have plans to use Application Deployment policy, ensure we select one among the three (Rolling, Rolling with Additional batch or Immutable) during environment creation.
  - Important Note-2 & Tip: After environment creation, we can always again switch back to "All at Once" if we want to and during that time Application Deployments section in Rolling Updates & Deployments will not be disappeared.

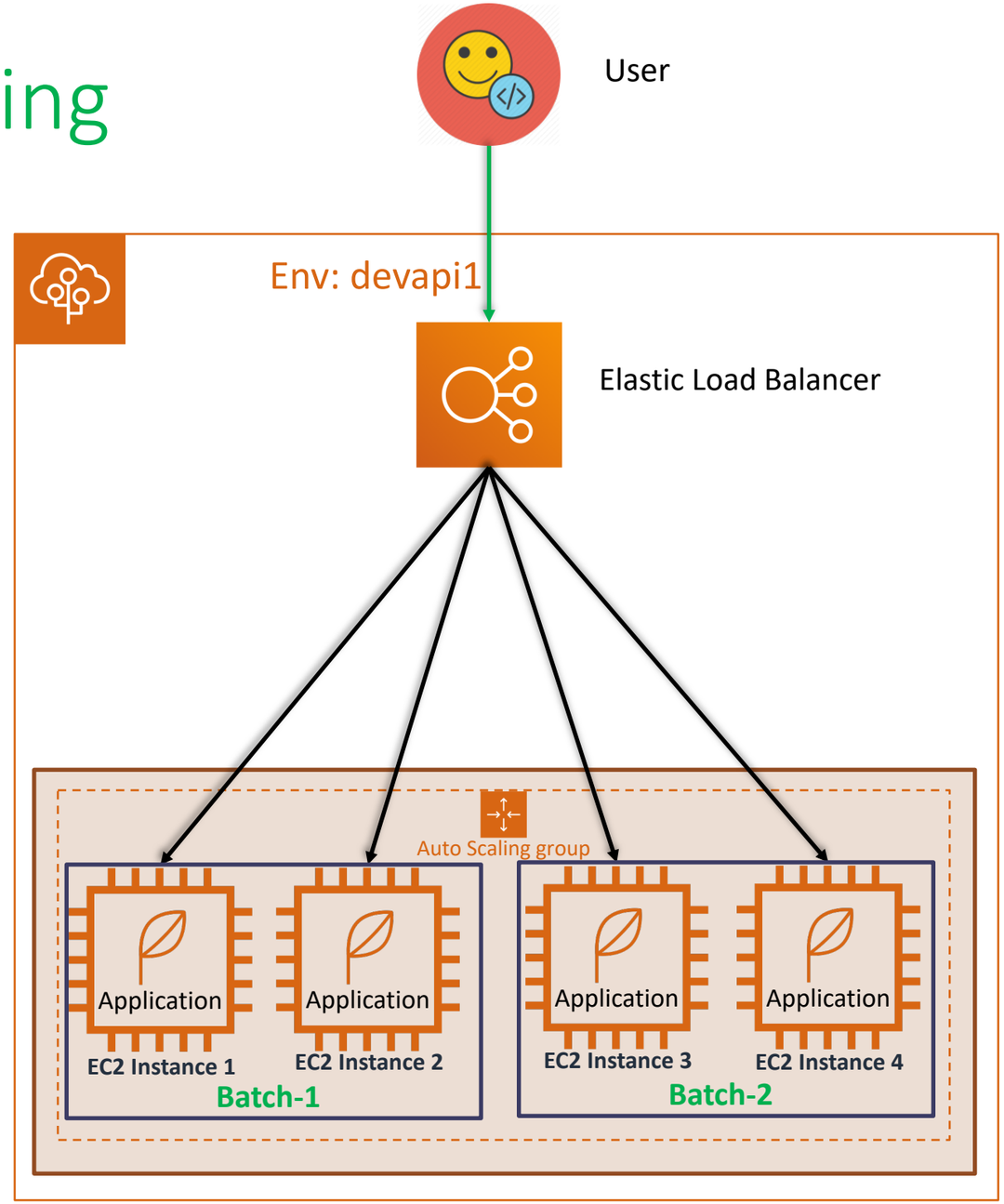
Application Deployments – All at once

- All at Once
  - By default, environment uses all-at-once deployments.
  - We will have a Service Disruption during the deployment
  - This is not a recommended option for application deployments if Availability of application is a primary requirement.



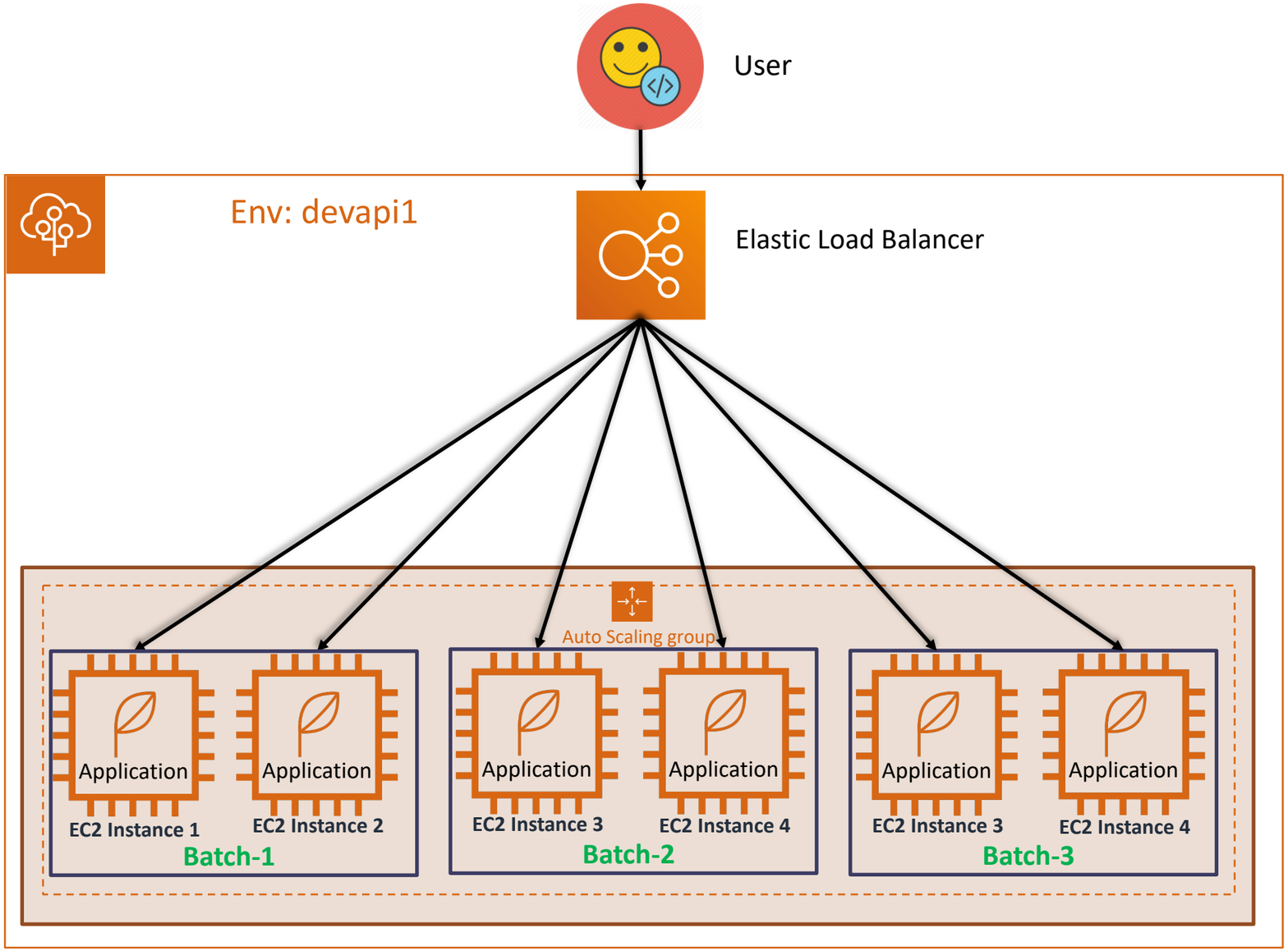
Application Deployments – Rolling

- Rolling Deployments
  - Elastic Beanstalk splits the environment's EC2 instances into batches and deploys the new version of the application to one batch at a time, leaving the rest of the instances in the environment running the old version of the application.
  - During a rolling deployment, some instances serve requests with the old version of the application, while instances in completed batches serve other requests with the new version.
  - Flow
    - Detach a batch from LB
    - Deploy new version
    - Re-attach the batch to LB
    - Wait for ELB health checks to pass
    - Once health check passed start routing requests.
    - Proceed with next batch



Application Deployments – Rolling with additional batch

- Rolling Deployments with additional batch
  - To maintain full capacity during deployments, we can configure our environment to launch a new batch of instances before taking any instances out of service.
  - When the deployment completes, Elastic Beanstalk terminates the additional batch of instances.
- Flow
- New Batch (batch-3)
  - Launches a new batch (batch-3)of instances
  - Deploys new version of code during launch
  - Registers or attaches to ELB
  - Wait for ELB health checks to pass
- Existing Batch (1 or 2)
  - Picks another batch (1 or 2)
  - Detach a batch-1 from LB
  - Deploy new version
  - Re-attach the batch-1 to LB
  - Wait for ELB health checks to pass
- Left-Over Batch
  - Terminate batch-2 instances.



Application Deployments – Rolling with additional batch

- Step-1: Verify current version of application
  - http://<env-dns-url>/hello
- Step-2: Apply Deployment Policy – Rolling with additional batch.
- Step-3: Deploy new version V2 from applications page
  - Monitor the Events to understand what's happening
  - View Health & EC2 instances page for additional instances coming online
  - Access application after deployment
    - http://<env-dns-url>/hello

How Rolling Deployments work?

- Rolling Deployments – Basic Flow
  - Elastic Beanstalk detaches all instances in batch from load balancer
  - Deploys the new application version
  - Re-attaches the instances back to load balancer
  - Elastic Beanstalk waits till all instances in that batches are healthy before moving to next batch.
- Major Drawback
  - If a deployment fails after one or more batches completed successfully, the completed batches run new version of application while any pending batches will run with old version. Ends up with mixed application versions.
  - Solution: Manual intervention required to view each instances deployment ID on health page and terminate those instances with old version so that Elastic Beanstalk replaces those instances with new version of application.

Application Deployments – Additional Features

- Healthy Threshold
  - If our application doesn't pass all health checks, but still operates correctly at a lower health status, we can allow instances to pass health checks with a lower status, such as Warning, by modifying the **Healthy threshold** option.
- Ignore Health Check
  - If our deployments fail because they don't pass health checks and we need to force an update regardless of health status, specify the **Ignore health check** option.
- Rolling Restarts
  - When we specify a batch size for rolling updates, Elastic Beanstalk also uses that value for rolling application restarts.
  - We can use rolling restarts when we need to restart the proxy and application servers running on our environment's instances without downtime.
- Flow for Rolling Restarts
  - Batch #1: Detach from LB, Restart Application, Attache to LB, Wait for health checks to Pass
  - Batch #2: Detach from LB, Restart Application, Attache to LB, Wait for health checks to Pass

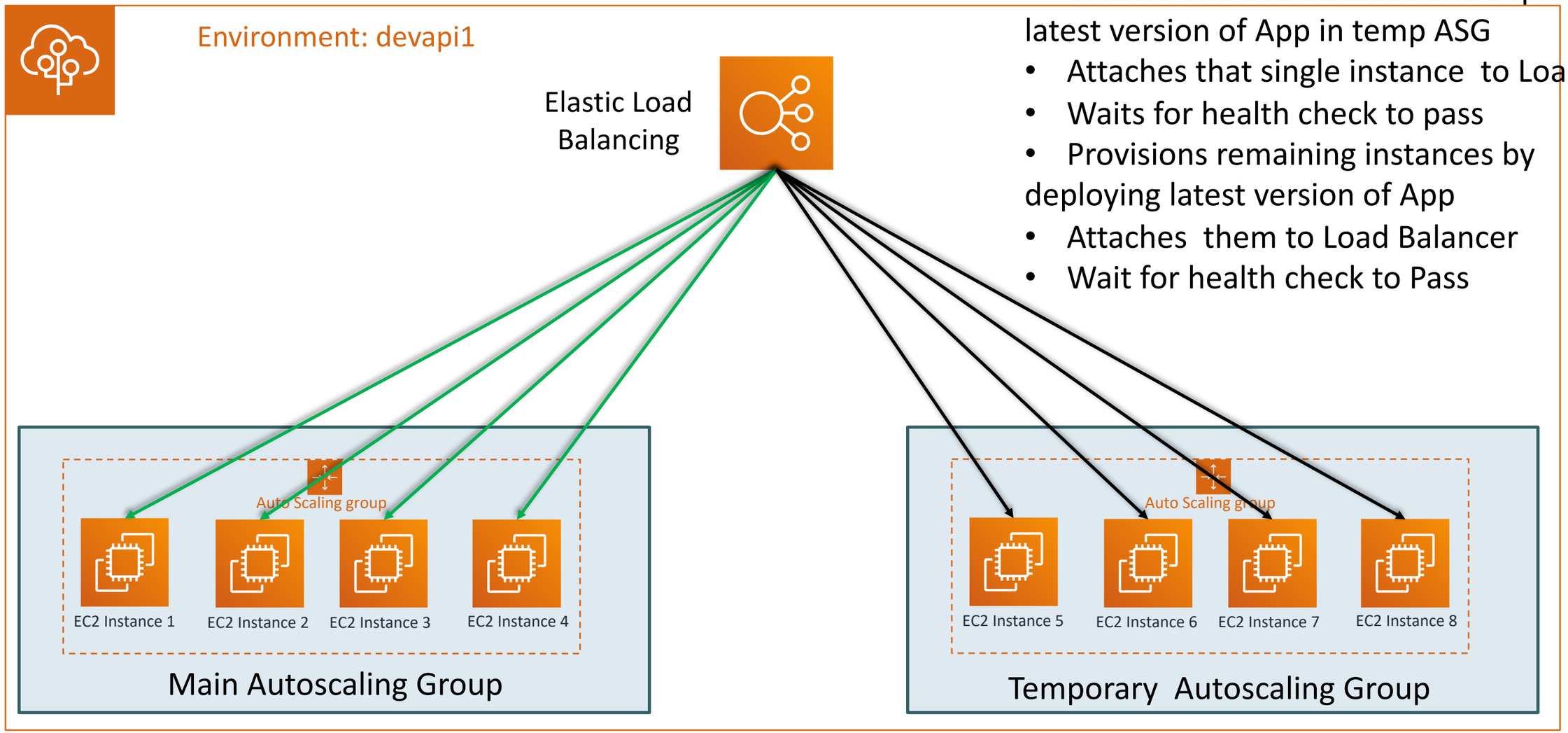
Application Deployments - Immutable

- Immutable Deployments
  - Immutable deployments perform an immutable update to launch a full set of new instances running the new version of the application in a separate Auto Scaling group, alongside the instances running the old version.
  - Immutable deployments can prevent issues caused by partially completed rolling deployments.
  - If the new instances don't pass health checks, Elastic Beanstalk terminates them, leaving the original instances untouched.

Application Deployments - Immutable

Immutable App Deployment Flow

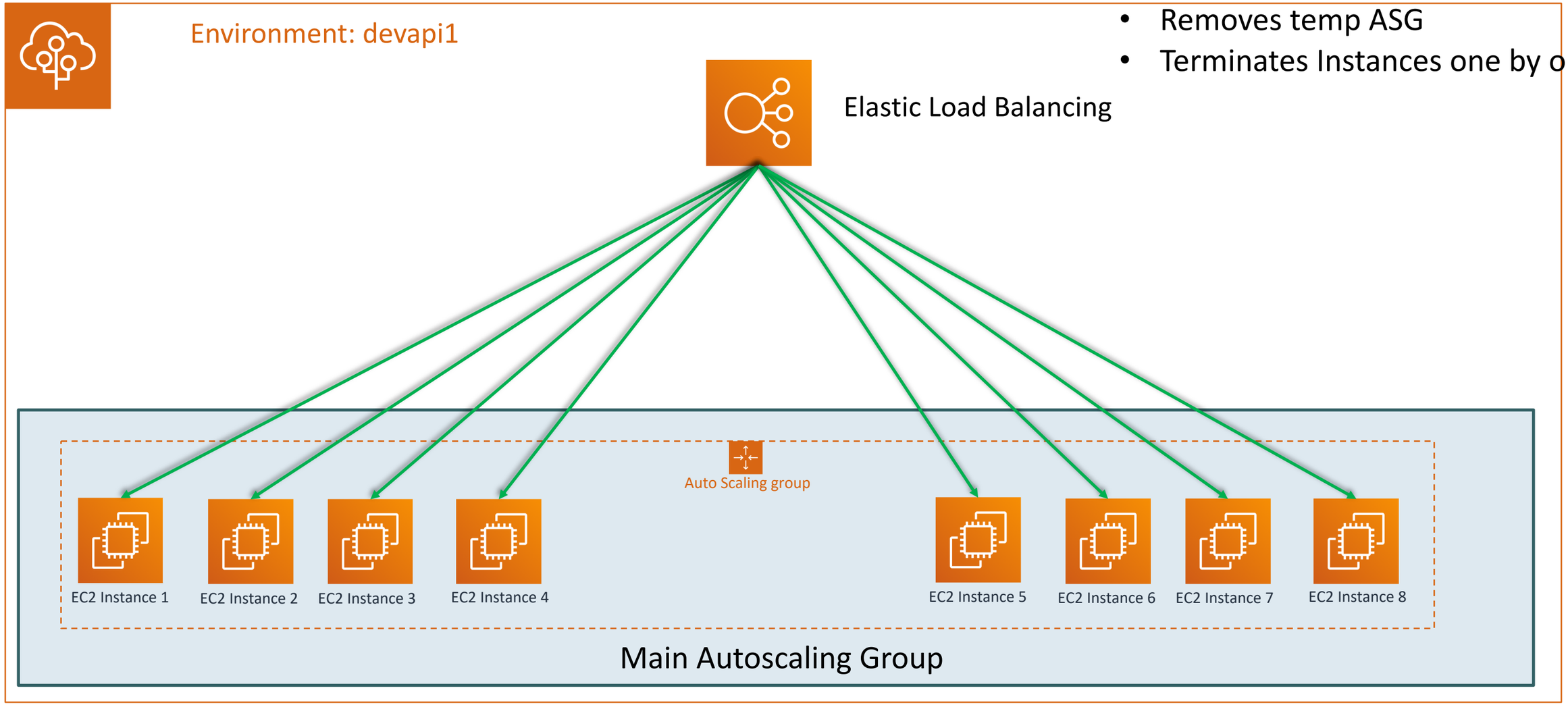
- Creates Temp Autoscaling group
- Creates one EC2 Instance and deploy latest version of App in temp ASG
- Attaches that single instance to Load balancer
- Waits for health check to pass
- Provisions remaining instances by deploying latest version of App
- Attaches them to Load Balancer
- Wait for health check to Pass



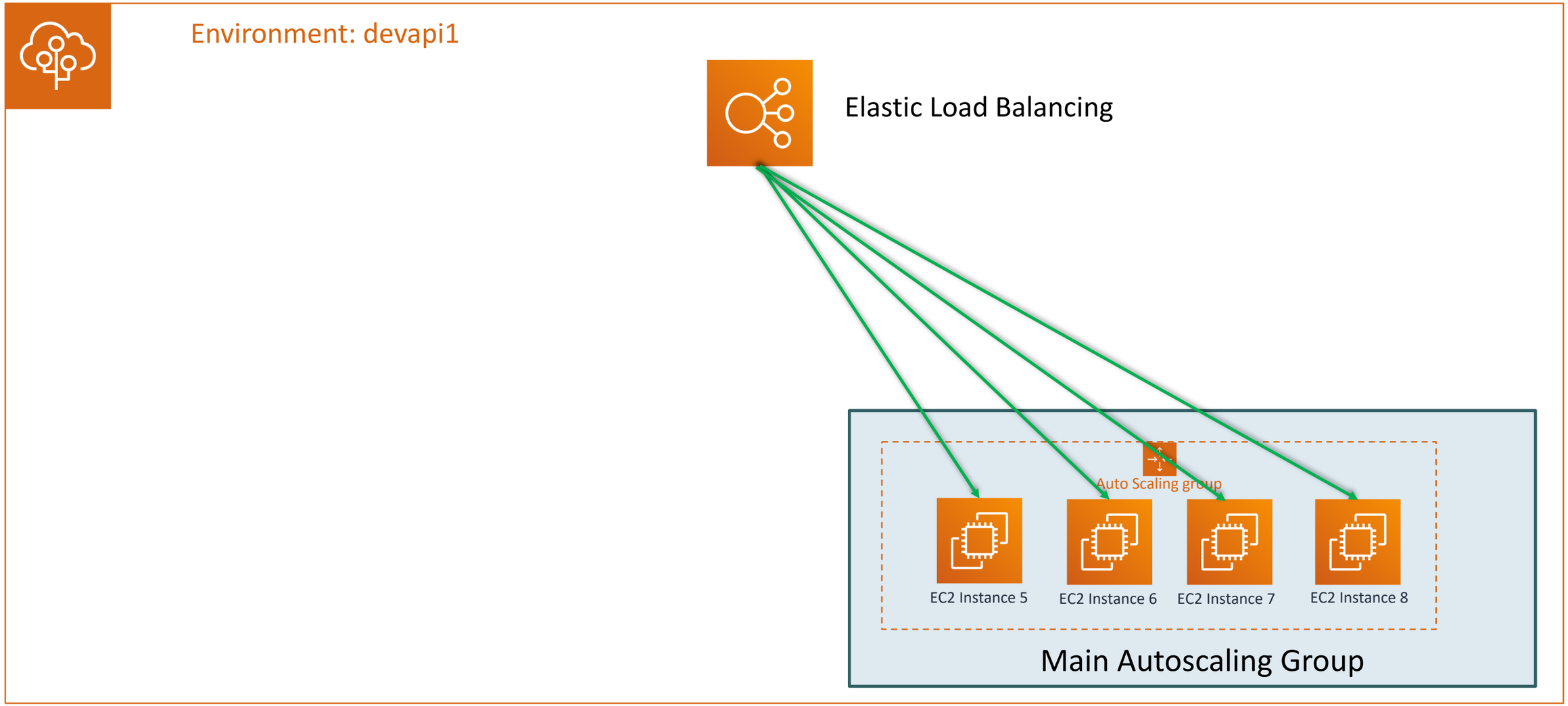
Application Deployments - Immutable

Immutable Application Deployments

- Adds new instances to main ASG
- Removes temp ASG
- Terminates Instances one by one.



Application Deployments - Immutable



Application Deployments - Immutable

- Step-1: Verify current version of application
  - http://<env-dns-url>/hello
- Step-2: Apply Deployment Policy – Immutable.
- Step-3: Deploy new version V3 from applications page
  - Monitor the Events to understand what's happening
  - View Health & EC2 instances page for additional instances coming online
  - Access application after deployment
    - http://<env-dns-url>/hello

Blue / Green Deployments

- Consider we have an environment named devapi16 running with version eb-app-v1

- If we want to deploy new version, the steps for Blue / Green Deployments include
  - Clone devapi16 and create new environment devapi17
  - Upload eb-app-v2 version to devapi17
  - Once all the tests are passed (health checks, app testing) then SWAP URLs for both environments.

- This is useful for smaller applications whose other aws services integration scope is small.

- Downside
  - If our environment has multiple integrations (DynamoDB, RDS DB, Elastic Cache, CICD pipelines and many other integrations) then cloning just an environment will not do.
  - Lot of other things need to be re-created for new environment including DB data to be in sync etc. Lot of permutations and combinations will come in to play for highly integrated applications, in that case Blue/Green deployments is not an option.
  - Creating new environment means its logs, analytics everything will be in new context so ideally not recommended in production.
  - Can be leveraged for multiple dev, qa and staging environments with different versions deployed so that we can SWAP urls as per need.

Which is the best Application Deployment Option?

Elastic Beanstalk Environment

Application Deployments

- **All at Once**
  - Service disruption during deployments
- **Rolling**
- **Rolling with additional batch**
  - In case of deployment failure, manual intervention is required
  - It ends up as partially completed deployment
- **Immutable**
  - Immutable deployments can prevent issues caused by partially completed rolling deployments.
  - If the new instances don't pass health checks, Elastic Beanstalk terminates them, leaving the original instances untouched.
- **Blue / Green Deployments**
  - This is useful for smaller applications whose other aws services integration scope is small.

Rolling Updates – Configuration Updates

Elastic Beanstalk Environment

Configuration Types

Replaces EC2 Instances

- Keypair change
- EC2 Instance Type change from t2.micro to something else
- Apply Managed Updates

No Impact to EC2 Instances

- Load Balancer Listener changes
- Load Balancer Health Status URL changes
- Environment Properties addition
- Add Notifications

Rolling Updates - Configurations

Elastic Beanstalk Environment

Configuration Updates (Rolling Update Type)

Disabled

- Updates applied all at once
- Instance replacement will be triggered
- Service Disruption

Rolling based on Health

- Updates happen in batches
- Minimum number of instances will be in service all times
- Moves to next batch once the current bacth health check passes

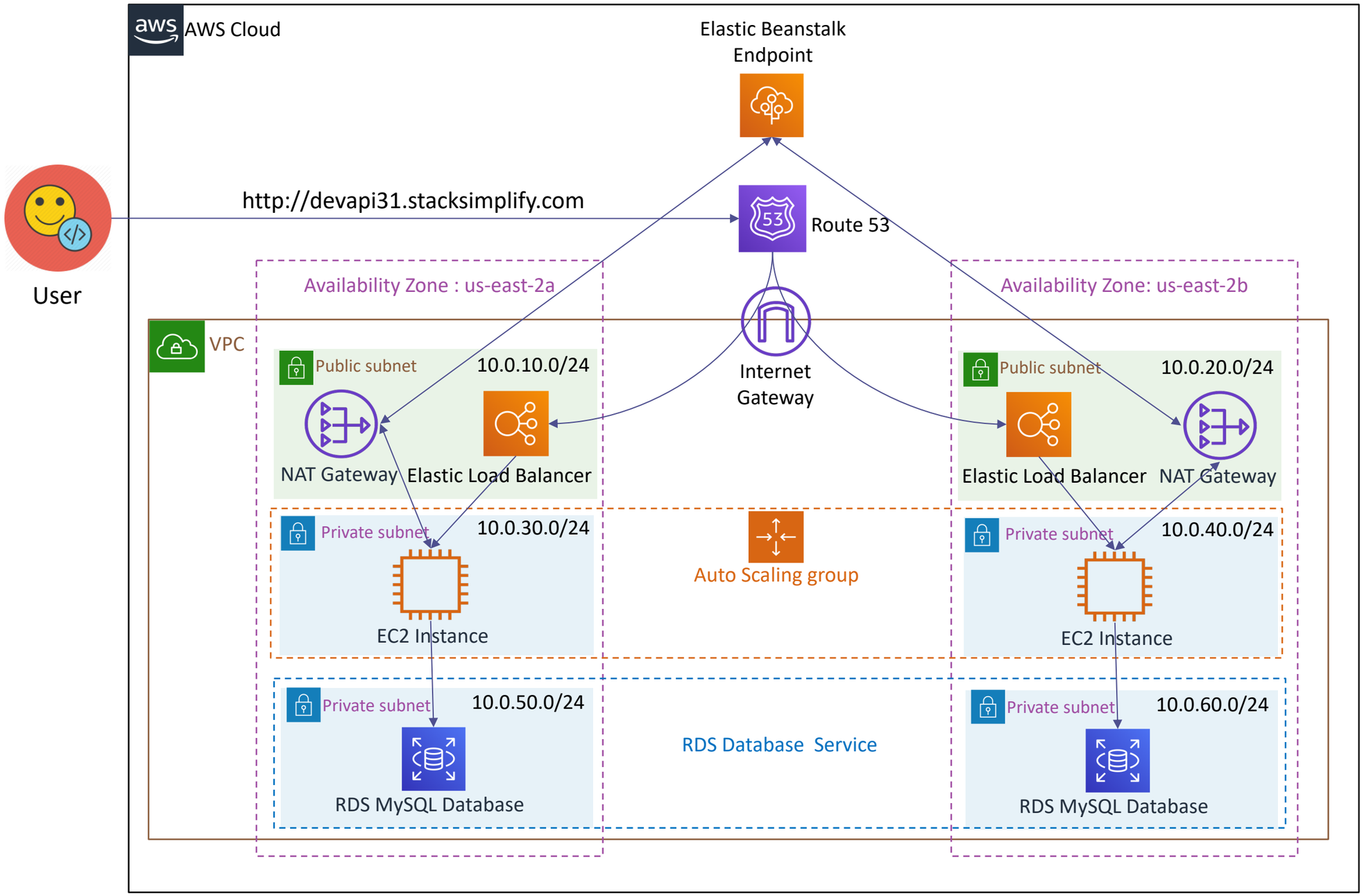
Rolling based on Time

- Updates happen in batches
- Minimum number of instances will be in service all times
- Moves to next batch once the Pause Time reaches

Immutable

- Same as Immutable Application Deployments

Elastic Beanstalk
&
Virtual Private Cloud (VPC)
&
Relational Database Service
(RDS)



Public Routes	
Destination	**Target**
10.0.0.0/16	local
0.0.0.0/0	IGW

Private Instance 2a Routes	
Destination	**Target**
10.0.0.0/16	local
0.0.0.0/0	NAT-2a-GW

Private Instance 2b Routes	
Destination	**Target**
10.0.0.0/16	local
0.0.0.0/0	NAT-2b-GW

Elastic Beanstalk - Network & Database

- Step-1: Create VPC Network
  - Create VPC
  - Create Subnets
  - Create Internet Gateway & associate to VPC
  - Create Route Tables

VPC Name	CIDR Block
VPC2	10.0.0.0/16

Subnet Name	Subnet IP Block	Availability Zone
public-2a-elb	10.0.10.0/24	us-east-2a
public-2b-elb	10.0.20.0/24	us-east-2b
private-2a-instance	10.0.30.0/24	us-east-2a
private-2b-instance	10.0.40.0/24	us-east-2b
private-2a-database	10.0.50.0/24	us-east-2a
private-2b-database	10.0.60.0/24	us-east-2b

Public Routes	
Destination	**Target**
10.0.0.0/16	local
0.0.0.0/0	IGW

Internet Gateway Name	VPC Name to be associated
vpc2-igw	VPC2

Route Table Name	VPC Name	Additional Routes
vpc2-public-routes	vpc2	Add Internet Route via vpc2-igw
vpc2-private-2a-routes	vpc2	Associate NAT-2a
vpc2-private-2b-routes	vpc2	Associate NAT-2b

VPC Design

VPC Size	Netmask	Subnet Size	Hosts/Subnet*	Subnets/VPC	Total IPs*
Micro	/24	/27	27	8	216
Small	/21	/24	251	8	2008
Medium	/19	/22	1019	8	8152
Large	/18	/21	2043	8	16344
Extra Large	/16	/20	4091	16	65456

Documentation Reference: https://aws.amazon.com/answers/networking/aws-single-vpc-design/

Elastic Beanstalk - Network & Database

- Step-2: Create two NAT Gateway
  - Create two Elastic IPs
  - NAT Gateway 1
    - Create NAT Gateway in public Subnet 2a
    - Create routes in private route table 2a to route outbound traffic via NAT Gateway NAT-2a
  - NAT Gateway 2
    - Create NAT Gateway in public Subnet 2b
    - Create routes in private route table 2b to route outbound traffic via NAT Gateway NAT-2b

NAT Gateway Name	Subnet	Elastic IP Allocation ID
NAT-2a	public-2a-elb	Select EIP
NAT-2b	public-2b-elb	Select EIP

Route Table Name	VPC Name	Additional Routes
vpc2-private-2a-routes	vpc2	Associate NAT-2a
vpc2-private-2b-route	vpc2	Associate NAT-2b

Private Instance 2a Routes	
Destination	**Target**
10.0.0.0/16	local
0.0.0.0/0	NAT-2a-GW

Private Instance 2b Routes	
Destination	**Target**
10.0.0.0/16	local
0.0.0.0/0	NAT-2b-GW

Elastic Beanstalk - Network & Database

- Step-3: Understand the Spring Boot Application
  - Understand GIT branches
    - 01-Unprotected-H2
    - 02-Protected-MySQL
    - master
  - Understand User Management Application Packages
    - User Controller
    - Admin User Controller
    - Application Status Controller
    - Hello World Controller
    - Authorization Configuration (OAuth)
    - pom.xml (jar file name)
  - Understand application.properties
    - application-h2.properties
    - application-mysqlaws.properties
    - application.properties

This is an Optional Lecture for System Admins but good to learn.

Elastic Beanstalk - Network & Database

- Step-4: Build, Package & Upload to Elastic Beanstalk
  - Build & Package Spring Boot Application
  - Upload the jar file to Elastic Beanstalk Application Versions page.
  - You can even directly download the jar file from "EB-Applications-Jars/RDS-MySQL" folder from course artifacts.

Elastic Beanstalk - Network & Database

- Step-5: Create Environment
  - High Availability
    - Select high availability for this environment
  - Database Config
    - Username: dbadmin1
    - Password: dbpassword1
  - Network Config
    - Select public subnets for ELB
    - Select private instance subnets for Instances
    - Select private database subnets for databases
  - Security Config
    - Add Key Pair
- Important Note: If NAT gateway route not configured in private subnets, elastic beanstalk and ec2 instances communication will not be established and environment creation will fail.

Elastic Beanstalk - Network & Database

- Step-6: Update Configuration post environment creation
  - 502 Bad Gateway Error
    - We get 502 error because application not able to connect to database because it doesn't have database information
  - Update Database Environment Variables
    - Collect Database information from RDS
    - Add database environment variables

    in Configuration → Software
    - Restart Application Server
    - Access application & verify logs(if required)

Variable Name	Variable Value
AWS_RDS_HOSTNAME	aa1ecc24l3hfakf.cjskxg02a3pt.us-east-2.rds.amazonaws.com
AWS_RDS_PORT	3306
AWS_RDS_DB_NAME	ebdb
AWS_RDS_USERNAME	dbadmin1
AWS_RDS_PASSWORD	dbpassword1

Elastic Beanstalk - Network & Database

- Update Health Monitor
  - Update load balancer health monitor URI (/health/status) and response code 200.
- Environment should be healthy after above two changes

Elastic Beanstalk - Network & Database

- Step-7: Create Admin User
  - Create Admin User in user management application for OAuth Token generation to access other APIs
  - Important Note: Admin User creation API created and unprotected for convenience but in real world we provision admin user directly to database.
  - Test all services
- Step-8: DNS Register the EB Environment URL
  - Route53
    - Create a registered domain (if not created)
    - Go to Hosted Zones
    - Create record set
    - Test all services with DNS registered url

Key	Value
Name	api.stacksimplify.com
Type	A-IPv4 address
Alias	Yes
Alias Target	Select EB environment URL

Elastic Beanstalk & RDS Database - Options

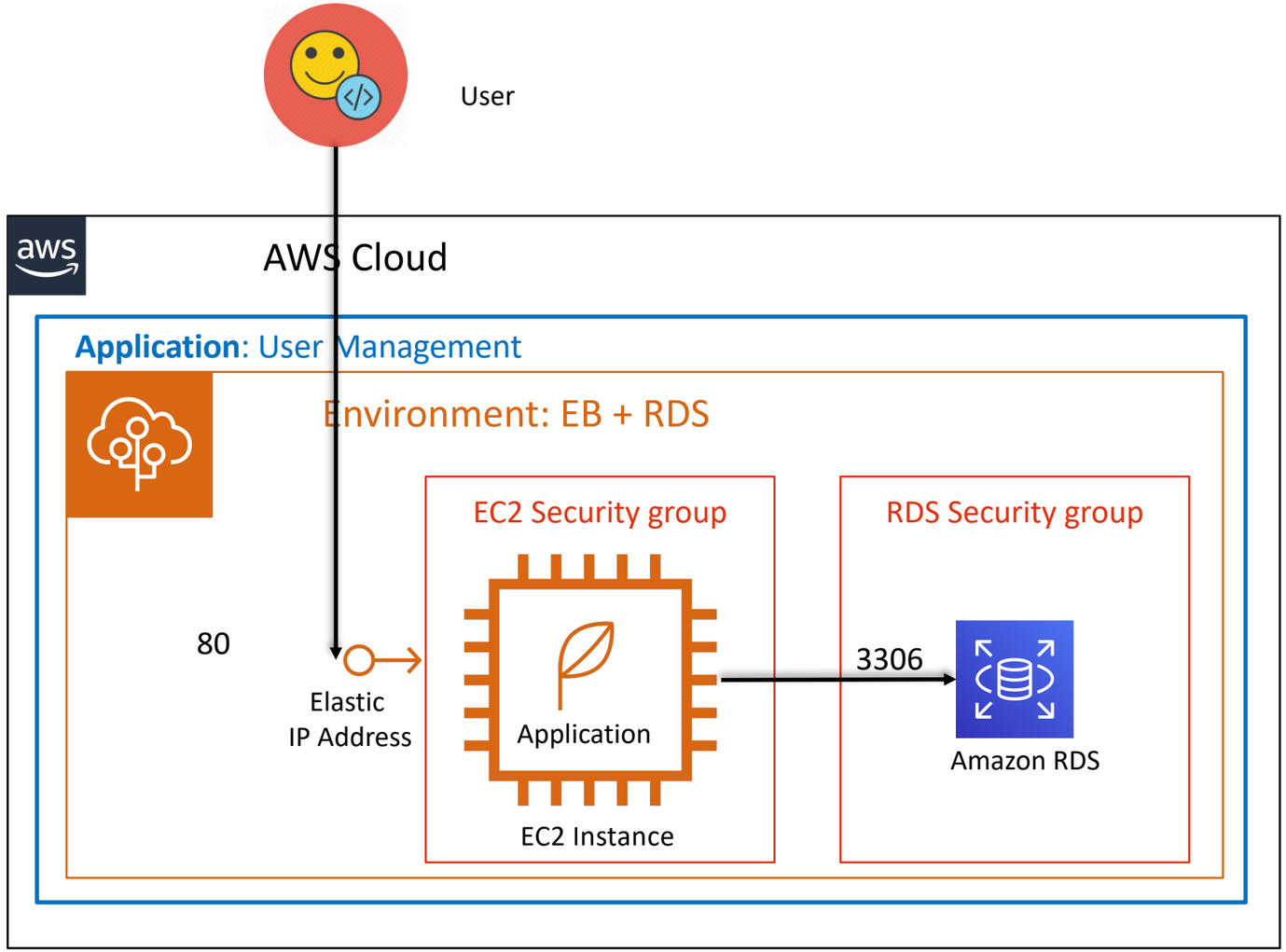
- Option-1: Create RDS database as part of Elastic Beanstalk environment
- Option-2: Create RDS database separately independent of Elastic Beanstalk environment.

Elastic Beanstalk
&
Relational Database Service
(Externalized)

Option-1: RDS Database part of Elastic Beanstalk

RDS DB part of Elastic Beanstalk

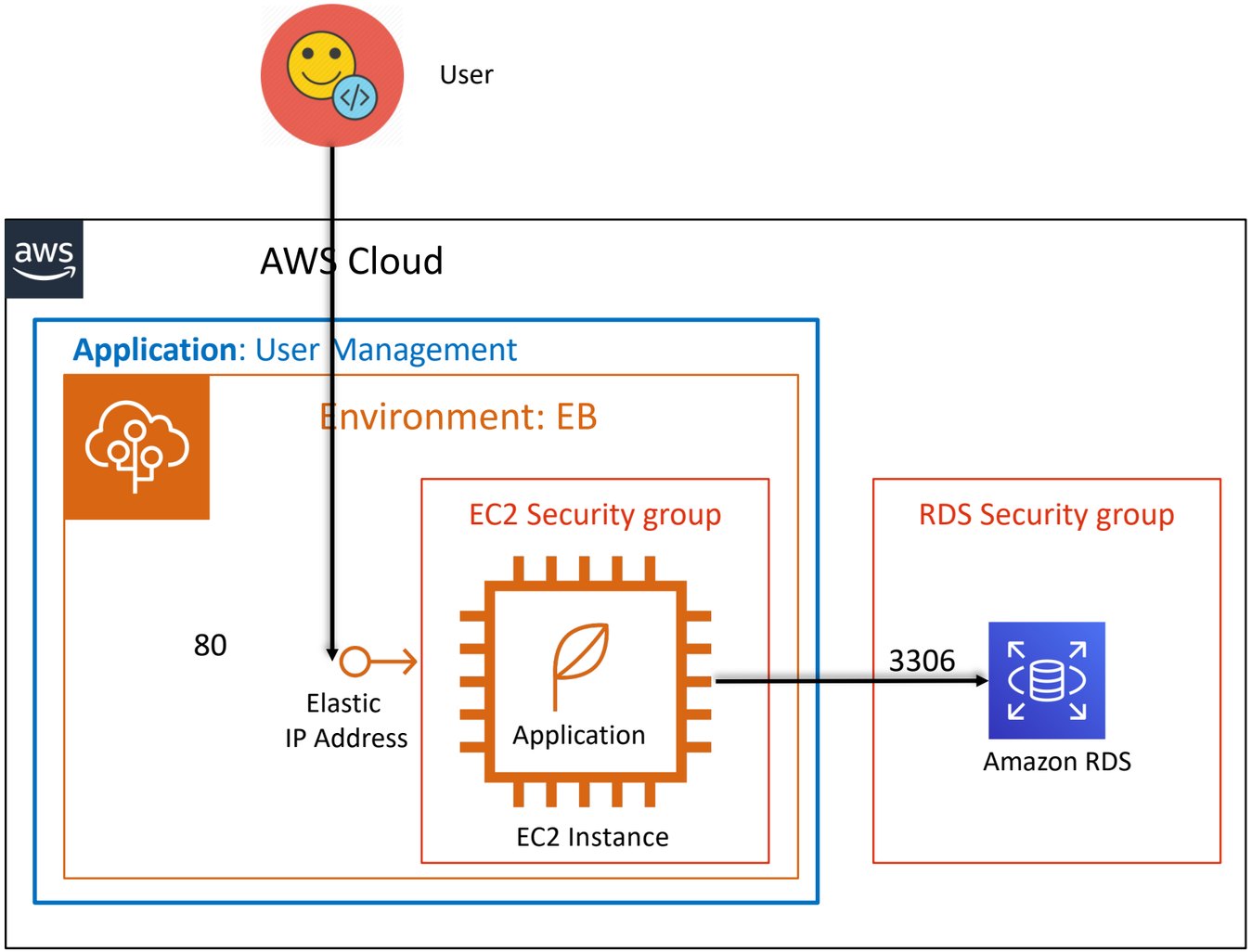
- Cost: cost is going to be high if we have 1 RDS db per environment and if we have multiple environments
- Not recommended for production environments if we delete EB environment Database gets deleted
- Useful for quick creation and termination environments (temp environments)



Option-2: RDS Database external to Elastic Beanstalk

RDS DB external to Elastic Beanstalk

- Cost: cost will be reduced as we created multiple environment schemas in single RDS database (Primarily for Dev, QA and Staging environments).
- Highly recommended for production environments because RDS DB in this case acts as independent database.
- Useful for permanently running environments by connecting to externalized database.



EB & RDS Combination – Pros, Cons

RDS DB part of Elastic Beanstalk

- Cost: cost is going to be high if we have 1 RDS db per environment and if we have multiple environments
- Not recommended for production environments if we delete EB environment Database gets deleted
- Useful for quick creation and termination environments (temp environments)

RDS DB external to Elastic Beanstalk

- Cost: cost will be reduced as we created multiple environment schemas in single RDS database (Primarily for Dev, QA and Staging environments).
- Highly recommended for production environments because RDS DB in this case acts as independent database.
- Useful for permanently running environments by connecting to externalized database

Option-2: Create RDS database separately independent of Elastic Beanstalk environment.

- Step-1: Create RDS Database
  - Create RDS Database
    - Pre-requisite: default VPC should be present in that region where you are creating this database with below options (below are minimal options with which we can create independent database)
    - Database Creation Method: Easy Create
    - Configuration: MySQL
    - DB Instance Size: Free Tier
    - DB Instance Identifier: usermgmtdb1
    - Master Username: dbadmin1
    - Master Password: dbpassword1
- Step-2: Update Database security group
  - Rule: Inbound
    - Type: MySQL/Aurora
    - Protocol: TCP
    - Porta Range: 3306
    - Source: My IP
    - Description: Access RDS database from local desktop to create “usermgmt” schema

Option-2: Create RDS database separately independent of Elastic Beanstalk environment.

- Step-3: Connect to MySQL DB and create schema
  - Install mysql workbench on your local desktop
  - Connect to database
    - Hostname: gather from database connectivity & security tab (Endpoint field)
    - Username: dbadmin1
    - Password: dbpassword
    - In SQL editor execute mysql query “create database dev33usermgmt;”
- Step-4: Create Environment
  - Pre-requisite: default VPC should be present in that region where you are creating this environment (or) custom VPC same as earlier depiction network and database section.
  - Environment Name: devapi33
  - Environment Tier: Webserver Environment
  - Environment Type: Single Instance / Load Balanced
  - Platform: Java
  - Application Code: Existing Code (eb-usermgmt-mysql-v1)
  - Create Environment

Option-2: Create RDS database separately independent of Elastic Beanstalk environment.

- Step-5: Update Database security group
  - Add new Rule: Inbound
    - Type: MySQL/Aurora
    - Protocol: TCP
    - Porta Range: 3306
    - Source: <Elastic Beanstalk environment security group>
    - Description: Requests coming from elastic beanstalk environment should be allowed by database security group to connect to database.
- Step-6: Update Database Info in EB
  - Update Database Environment Variables
    - Collect Database information from RDS
    - Add database environment variables

    in Configuration → Software
    - Access application & verify logs(if required)

Variable Name	Variable Value
AWS_RDS_HOSTNAME	usermgmtdb1.cjskxg02a3pt.us-east-2.rds.amazonaws.com
AWS_RDS_PORT	3306
AWS_RDS_DB_NAME	dev33usermgmt
AWS_RDS_USERNAME	dbadmin1
AWS_RDS_PASSWORD	dbpassword1

Option-2: Create RDS database separately independent of Elastic Beanstalk environment.

- Step-7: (Assignment) Create new environment devapi34
  - Connect to database and create new schema dev34usermgmt
  - Create Environment
  - Update Database security group with new environment
  - Update Database Environment Variables
    - Collect Database information from RDS
    - Add database environment variables

    in Configuration → Software
    - Access application & verify logs(if required)

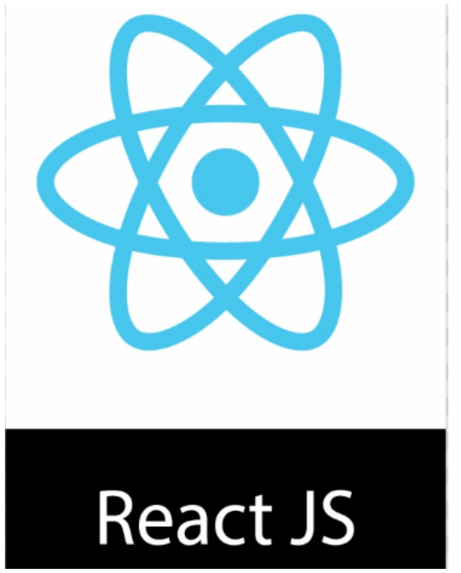
Variable Name	Variable Value
AWS_RDS_HOSTNAME	usermgmtdb1.cjskxg02a3pt.us-east-2.rds.amazonaws.com
AWS_RDS_PORT	3306
AWS_RDS_DB_NAME	dev34usermgmt
AWS_RDS_USERNAME	dbadmin1
AWS_RDS_PASSWORD	dbpassword1

AWS CloudFront

AWS S3 Static Sites

AWS Elastic Beanstalk

Full Stack Application Deployment



Spring Boot Restful API

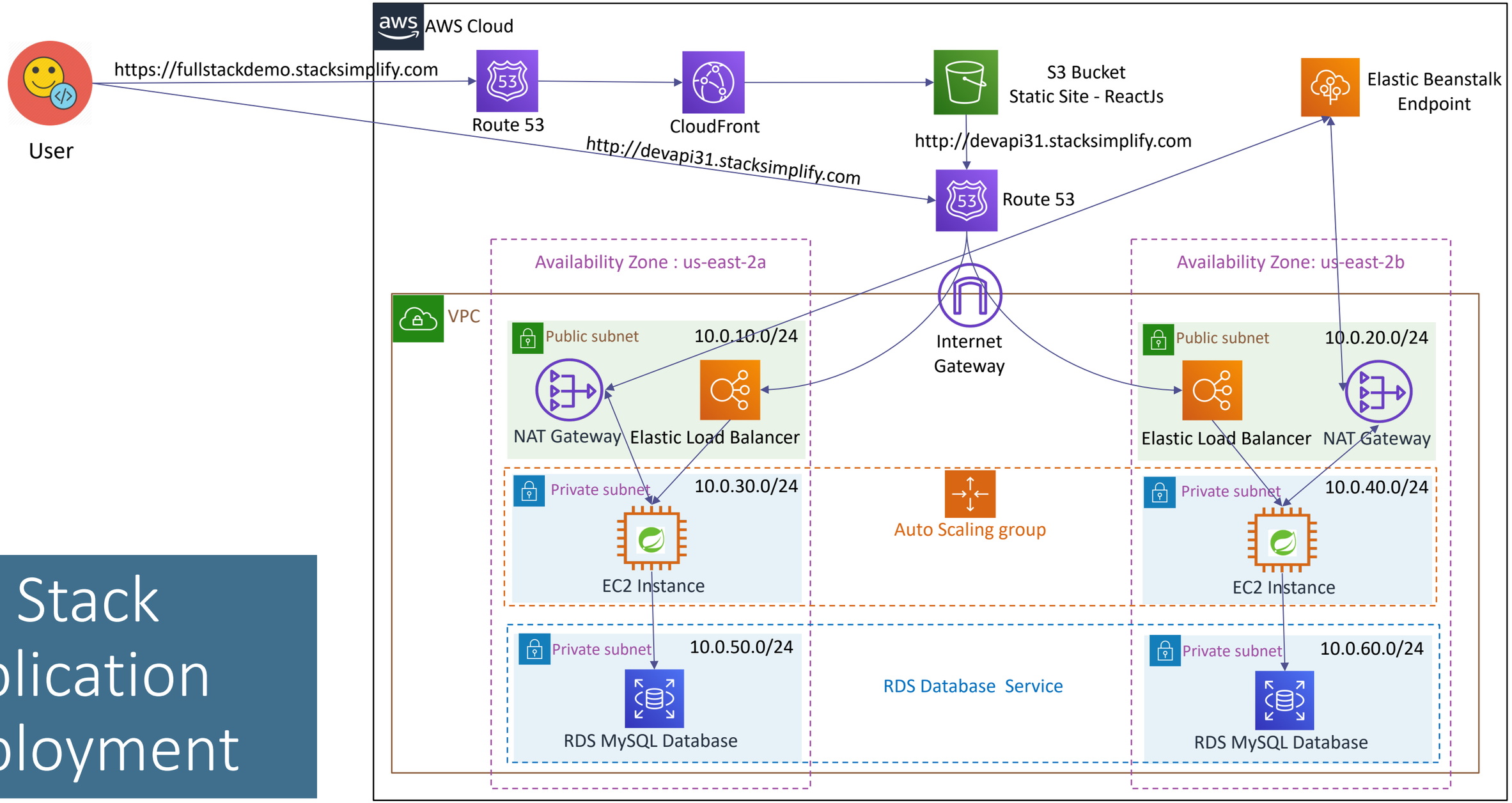
Why to learn Full Stack Deployments on AWS?

- Industry wide common and very high demand combination for building modern applications is Spring Boot & ReactJs
  - Backend: SpringBoot – RESTful API's
  - Frontend: ReactJs – Modern UI
  - Security: Authentication System – OAuth, JWT
- Learning full stack deployment on cloud providers and their DevOps usecases implementation will be a very high demand requirement today.
- In this section we will cover, full stack application deployments and in next section we will implement DevOps usecases (Continuous Integration & Continuous Delivery for Full Stack applications where-in our backend is deployed on Elastic Beanstalk.

ReactJs

- ReactJs is a javascript library for building user interfaces
- Very popular and highly used now a days.
- Refer the below link which contains the important trends about ReactJs (always on Top)
- https://medium.com/zerotomastery/tech-trends-showdown-react-vs-angular-vs-vue-61ffaf1d8706
- Google Trends URL:
- Which means in addition to ReactJs developers, there will be a demand for DevOps resources for managing ReactJs based applications on cloud platforms in combination with backend applications like spring boot.

Full Stack Application Deployment



Full Stack Application Deployment

- Step-1: ReactJS Application Test locally by pointing to local environment
- Step-2: ReactJS Application Test locally by pointing to Elastic Beanstalk Environment – devapi31
- Step-3: Setup Static Site on AWS S3 and upload ReactJs generated static content from build folder
- Step-4: Create CloudFront Distribution with SSL enabled
- Step-5: Route53 – Create Hosted zone with custom DNS

Full Stack Application Deployment

- Step-1: ReactJS Application Test locally by pointing to local SpringBoot environment
  - Install NodeJs on local desktop.
    - brew update
    - brew install node.
    - node –v
    - npm -v
  - Copy & Unzip ReactJs application from Course-Artifacts in folder “ReactJS-Frontend-App/ 02-eb-usermgmt-frontend-reactjs.zip”
  - Import project to VS Code Editor
  - Install Node Packages (Navigate to project folder and execute below command)
    - npm install
  - Verify the file .env.development
    - REACT_APP_USERMGMT_API_BASE_URL=http://localhost:5000
  - Test locally by pointing to local springboot environment
    - npm run start.

Full Stack Application Deployment

- Step-2: ReactJS Application Test locally by pointing to Elastic Beanstalk Environment
  - Understand the file .env.production and update Elastic Beanstalk environment endpoint.
  - Important Note: Remove “/” at the end of URL when updating in .env.production.
  - npm run build
  - npm install –g serve
  - serve –s build
  - Test ReactJS locally after executing above commands wherein API requests going to Elastic Beanstalk environment.
    - Verify using ReactJs UI (Create new user)
    - Verify using Postman ListUsers service pointing to devapi31 environment.

AWS S3 Buckets

- S3 stands for – Simple Storage Service
- S3 has a simple web services interface that you can use to store and retrieve any amount of data, at any time, from anywhere on the web.
- It gives any developer access to the highly scalable, reliable, fast, inexpensive data storage infrastructure that Amazon uses to run its own global network of web sites.
- Industry-leading performance, scalability, availability, and durability
- Wide range of cost-effective storage classes
- Unmatched security, compliance, and audit capabilities
- In short, most of AWS services use S3 as their underlying store including Elastic Beanstalk which we are discussing in this course.
- In upcoming sections, we will be implementing Continuous Integration & Delivery and you will see how S3 will be used there to store the build artifacts.

Full Stack Application Deployment

- Step-3: Static Site setup on AWS S3 and upload ReactJs project files
  - Create S3 Bucket
  - Make the bucket public
  - Add the bucket policy
  - Enable Static Website Hosting in bucket properties and make a note of S3 Endpoint
  - Upload Static content from **"build"** folder in ReactJs project folder.
  - Test the application using S3 endpoint noted from Static Website Hosting section.
  - Understand difference between regular S3 bucket endpoint & S3 endpoint for Static website hosting.

Bucket Policy

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "PublicReadGetObject",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::bucket-name/*"
    }
  ]
}
```

AWS CloudFront Introduction

- Speeds up distribution of static and dynamic web content
- CloudFront delivers our content through a worldwide network of data centers called edge locations.
- When a user requests content that we are serving with CloudFront, the user is routed to the edge location that provides the lowest latency (time delay), so that content is delivered with the best possible performance.
- If the content is already in the edge location with the lowest latency, CloudFront delivers it immediately.
- If the content is not in that edge location, CloudFront retrieves it from an origin that you've defined—such as an Amazon S3 bucket, or an HTTP server (for example, a web server) that we have identified as the source for the definitive version of our content.

Full Stack Application Deployment

- Step-4: Create CloudFront Distribution
  - Create Distribution of Type Web
  - Origin Domain Name: <Very important caution – provide S3 endpoint which is shown in Static Website hosting section of S3 bucket and not regular s3 endpoint>
    - demoapi31static.s3-website.us-east-2.amazonaws.com
  - Viewer Protocol Policy: HTTP & HTTPS
  - Alternate Domain Names: fullstackdemo.stacksimplify.com
  - SSL Certificate: *.stacksimplify.com
  - Discuss about other settings
  - Rest all settings leave to defaults
  - Wait for 10 to 15 minutes for distribution to get created and replicated across all edge locations.
  - Access the application using CloudFront domain name & test.

Full Stack Application Deployment

- Step-5: Route53 – Create Hosted zone with custom DNS
  - Make a note of CloudFront domain name for the distribution we created recently.
  - Create a Hosted Zone in Route53
    - Name: fullstackdemo.stacksimplify.com
    - Alias: Yes
    - Alias Target: <Copy cloudfront distribution domain name>
    - Click Create.
  - Wait for few minutes to changes to take place. Usually it takes 60 seconds for Route53 hosted zone propagations but sometimes it might even take 15to 20 minutes to clear DNS caches.
  - Access and test both HTTP and HTTPS urls
    - HTTP: http://fullstackdemo.stacksimplify.com
    - HTTPS: https://fullstackdemo.stacksimplify.com

CodeCommit

CodeBuild

Elastic Beanstalk

CodePipeline

CloudWatch

Simple Notification Service

Continuous Integration
&
Continuous Delivery

Stages in Release Process

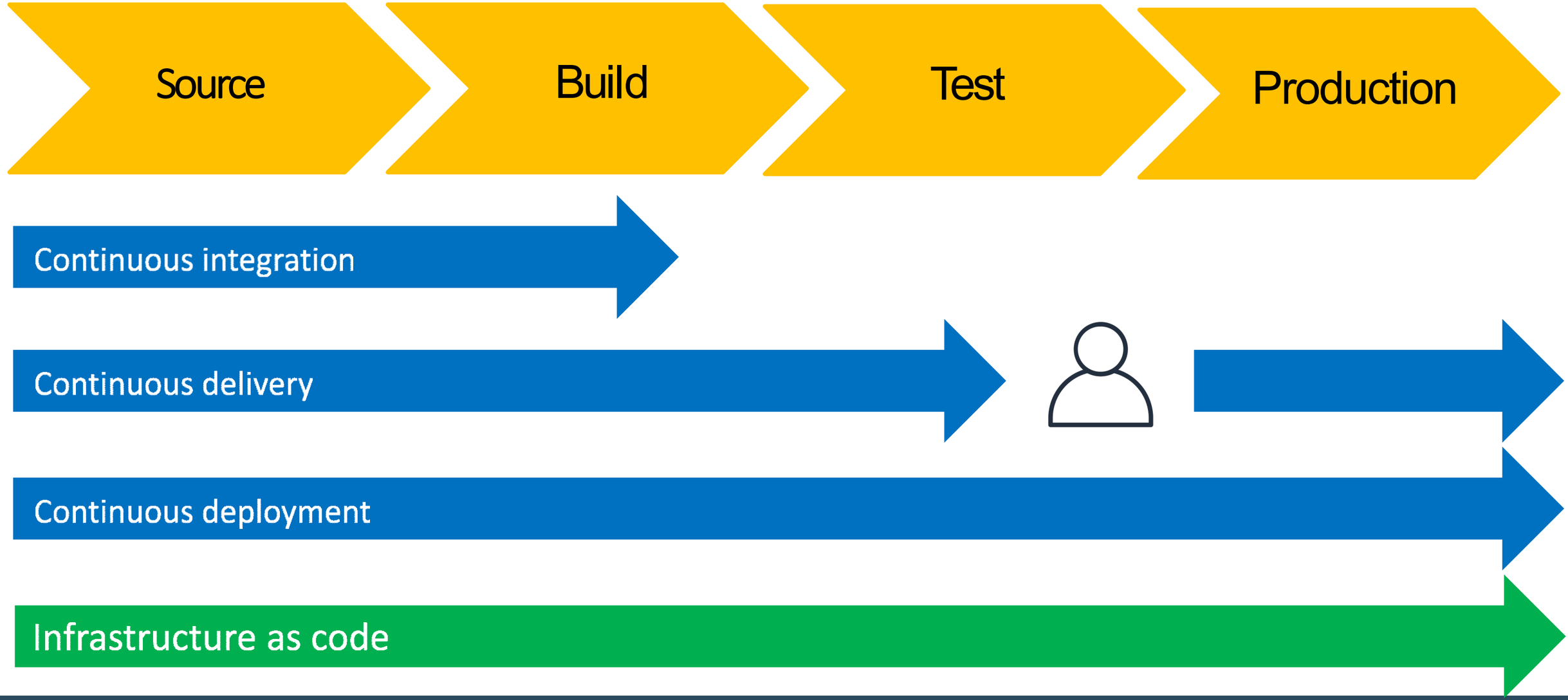
- Check-in source code
- Peer review new code
- Pull Request process

- Compile Code & build artifacts (war ,jar, container images, Kubernetes manifest files)
- Unit Tests

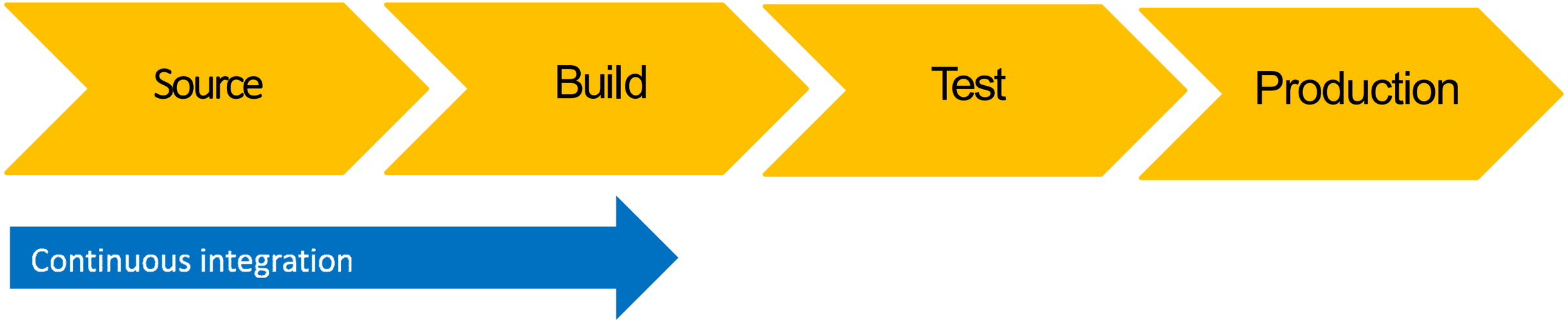
- Integration tests with other systems.
- Load Testing
- UI Tests
- Security Tests
- Test Environments (Dev, QA and Staging)

- Deployment to production environments
- Monitor code in production to quickly detect errors

Stages in Release Process



Continuous Integration

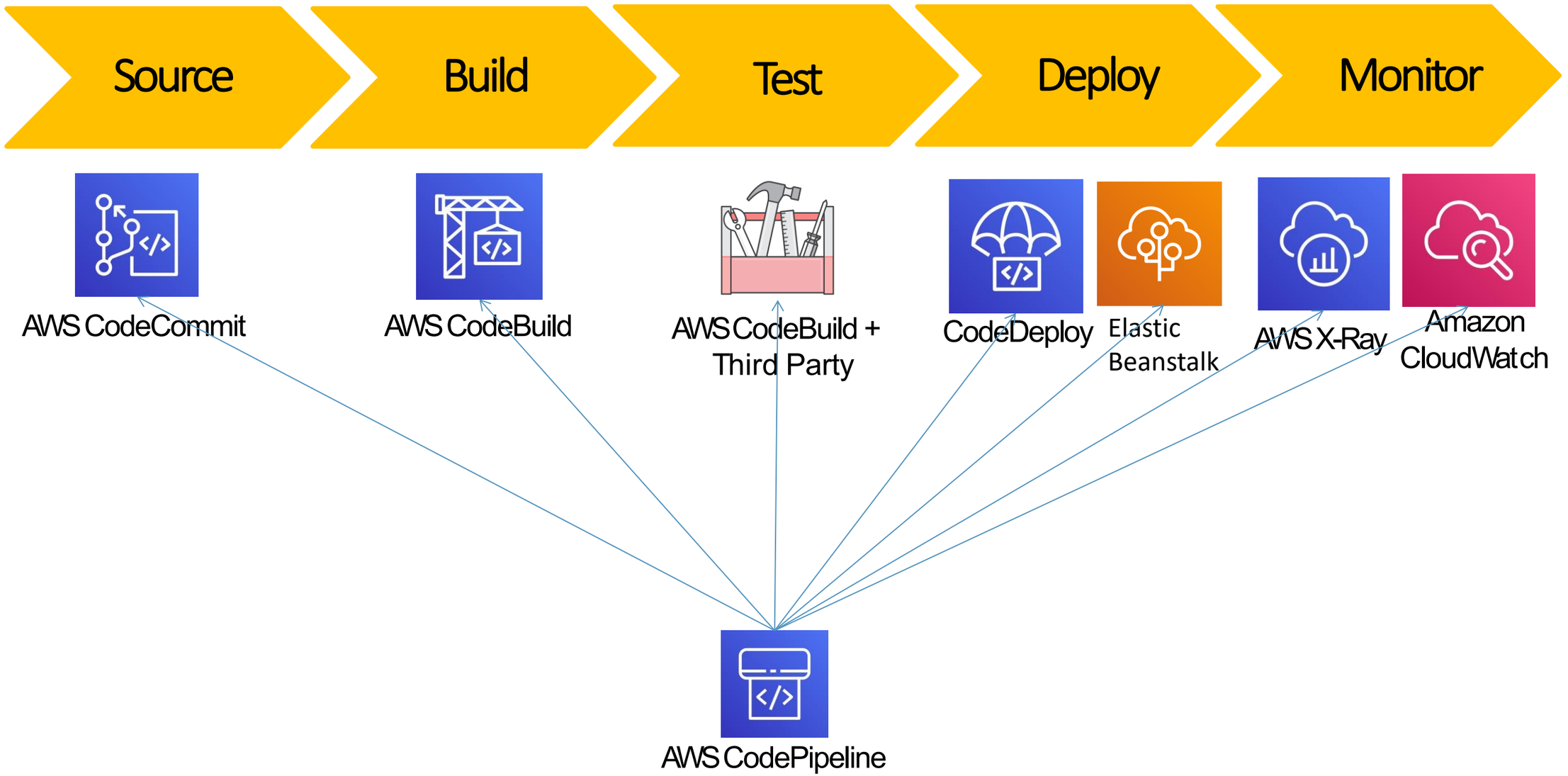


- Automatically kick off a new release when new code is checked-in
- Build and test code in a consistent, repeatable environment
- Continually have an artifact ready for deployment

Continuous Delivery

- Automatically deploy new changes to staging environments for testing
- Deploy to production safely without affecting customers
- Deliver to customers faster
- Increase deployment frequency, and reduce change lead time and change failure rate

AWS Developer Tools or Code Services



AWS Developer Tools or Code Services

AWS CodeCommit

AWS CodeBuild

AWS CodeBuild + Third Party

Elastic Beanstalk

AWS X-Ray

Amazon CloudWatch

AWS CodePipeline

AWS Developer Tools or AWS Code Services

Source

CodeCommit

- Version control service
- We can privately store and manage source code
- Secure & highly available

Build

CodeBuild

- Fully managed build service, Compiles source code, Runs tests and produces software packages
- Scales continuously and processes multiple builds concurrently.
- No build servers to manage.
- Pay by minute, only for compute resources we use.
- Monitor builds through CloudWatch events.
- Supports following programming language runtimes Ruby, Python, PHP, Node, Java, Golang, .Net Core, Docker and Android

Test

CodeBuild + Third Party

CodePipeline

- Continuous delivery service for fast and reliable application updates
- Model and visualize your software release process
- Builds, tests, and deploys your code every time there is a code change
- Integrates with third-party tools and AWS

Deploy

Elastic Beanstalk

- Automates code deployments to any instance of EB environment using EB container
- EB picks the artifacts generated by codeBuild and deploys to EB Environment

Monitor

AWS X-Ray

CloudWatch

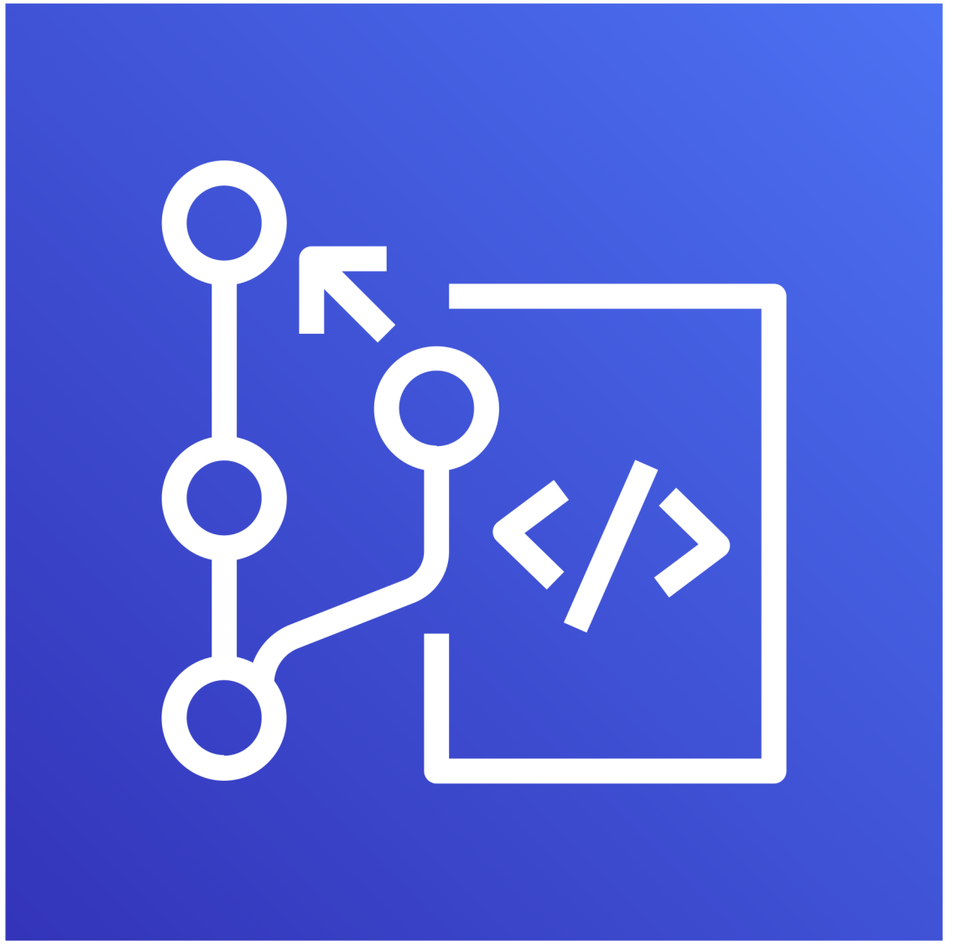
- Monitors Source check-ins and triggers builds
- Monitors builds
- Monitors Infrastructure
- Collects logs

Elastic Beanstalk
Pre-requisites for
CI CD Implementation

Elastic Beanstalk – Environment Creation for CICD

- Step-1: Create 3 environments for CICD implementation
  - Dev: leverage devapi31 created as part of Network & Database section (already exists)
  - Staging: Clone devapi31 and make necessary changes
  - Production: Clone devapi31 and make necessary changes
- Step-2: Staging environment – stageapi31
  - Clone devapi31 with name as stageapi31
  - For stageapi31, update RDS DB hostname in EB environment properties and test
- Step-3: Production Environment – prodapi31
  - Clone devapi31 with name as prodapi31
  - For prodapi31, update RDS DB hostname in EB environment properties and test

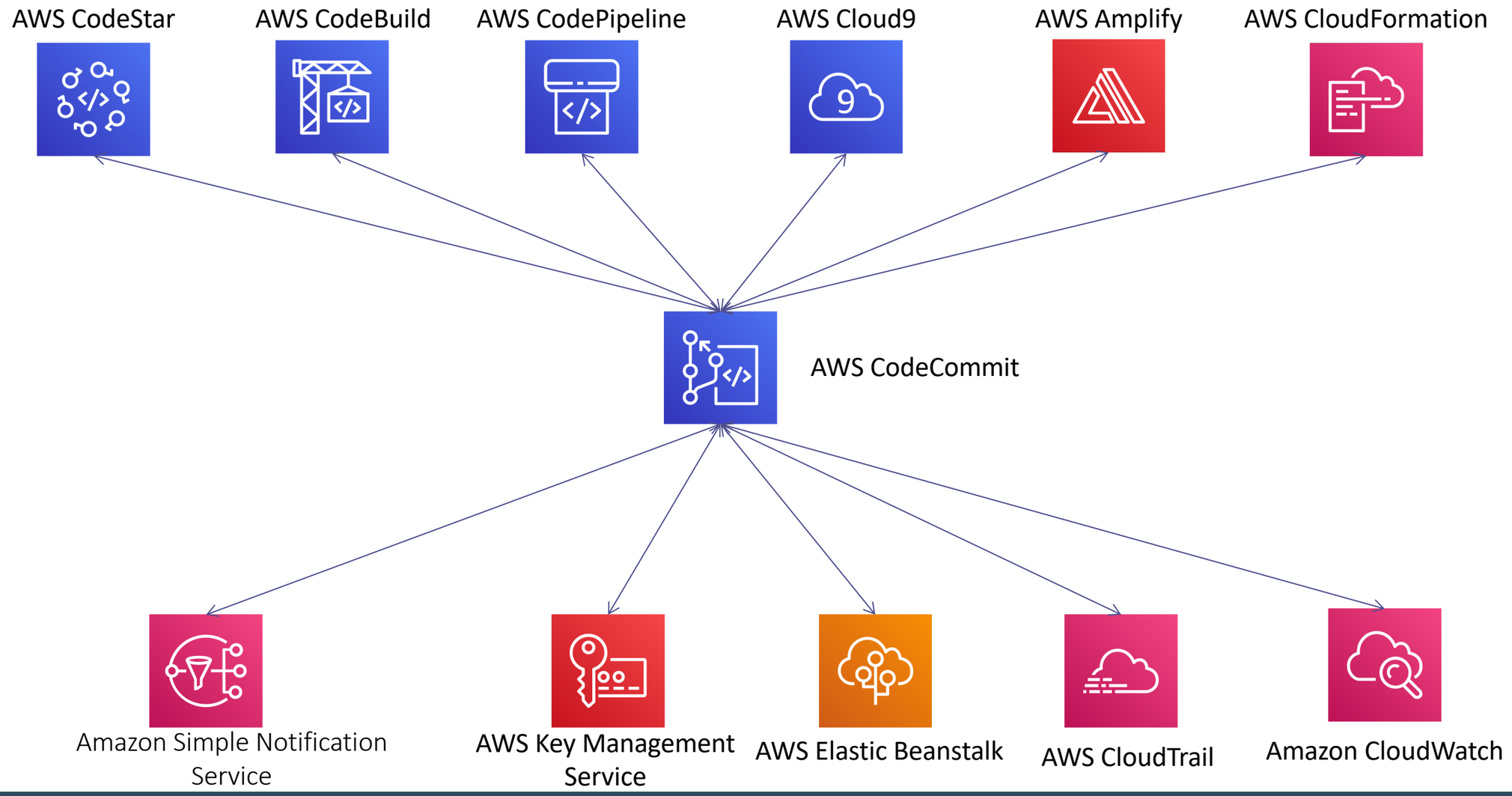
AWS CodeCommit



AWS CodeCommit - Introduction

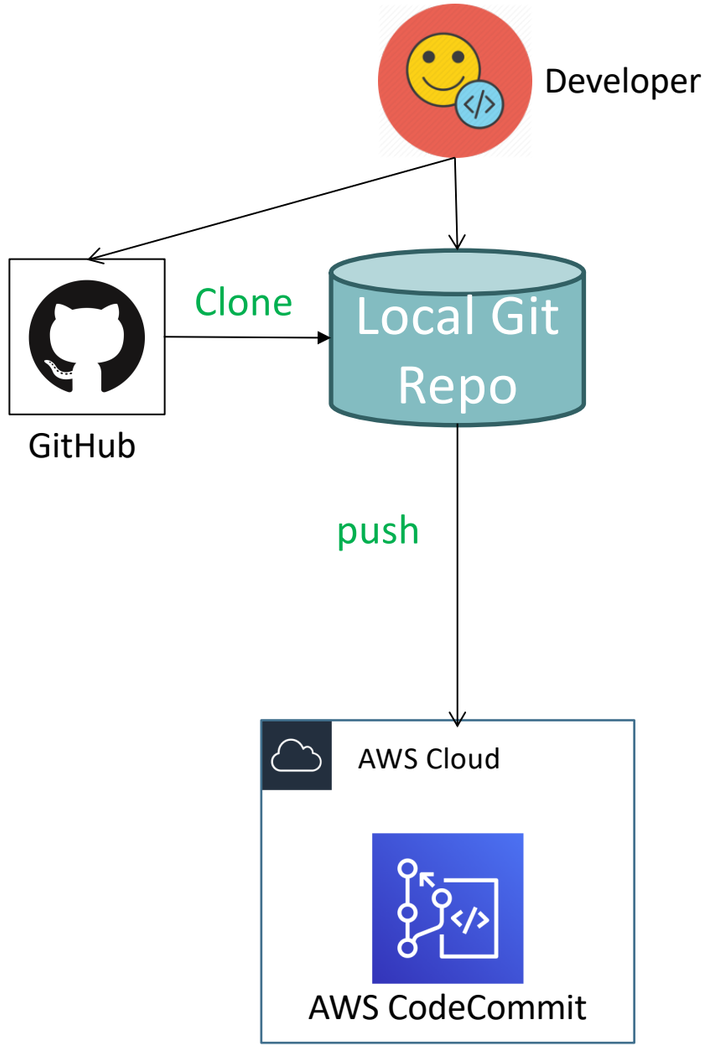
- Version Control Service hosted by AWS
- We can privately store and manage documents, source code, and binary files
- Secure & highly scalable
- Supports standard functionality of Git (CodeCommit supports Git versions 1.7.9 and later.)
- Uses a static user name and password in addition to standard SSH..

CodeCommit – Integration with AWS Services



CodeCommit - Steps

- Step-1: Project setup in Spring Tool Suite IDE
  - Pre-requisites: Install STS IDE
  - Clone the project 01-eb-usermgmt from https://github.com/stacksimplify/01-eb-usermgmt
  - Create local branches (3 branches – 01, 02 and master)
  - Delete Remotes→ origin
  - Run application locally once and test it (simple/ health/status api)
- Step#2: Remote GIT Repository
  - Create a remote git repository in AWS Code Commit.
  - Create Code Commit git credentials to connect.
  - Push the code to remote git repository.
  - Verify code in AWS Code Commit.
- Step#3: CodeCommit Features
  - Code, Commits, Branches
  - Settings: Notifications, Triggers
  - Pull Requests

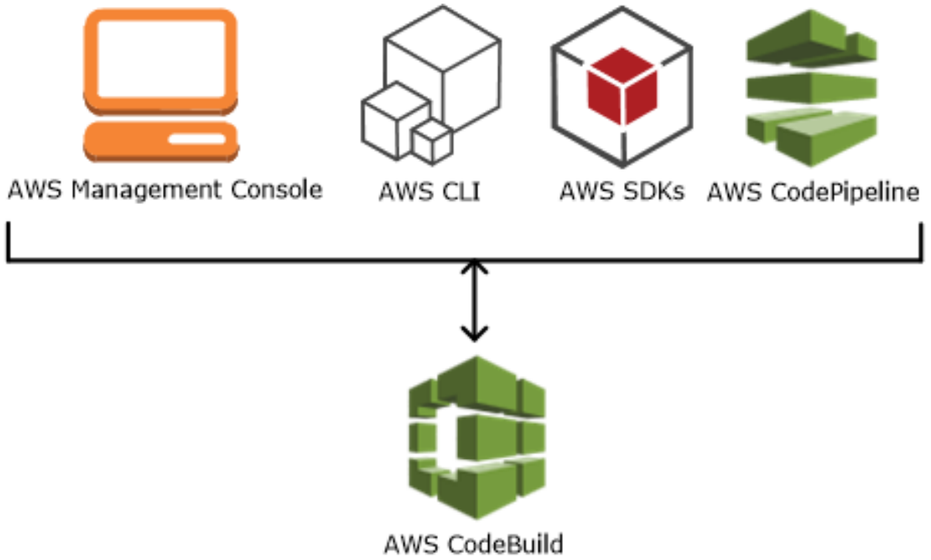


AWS CodeBuild

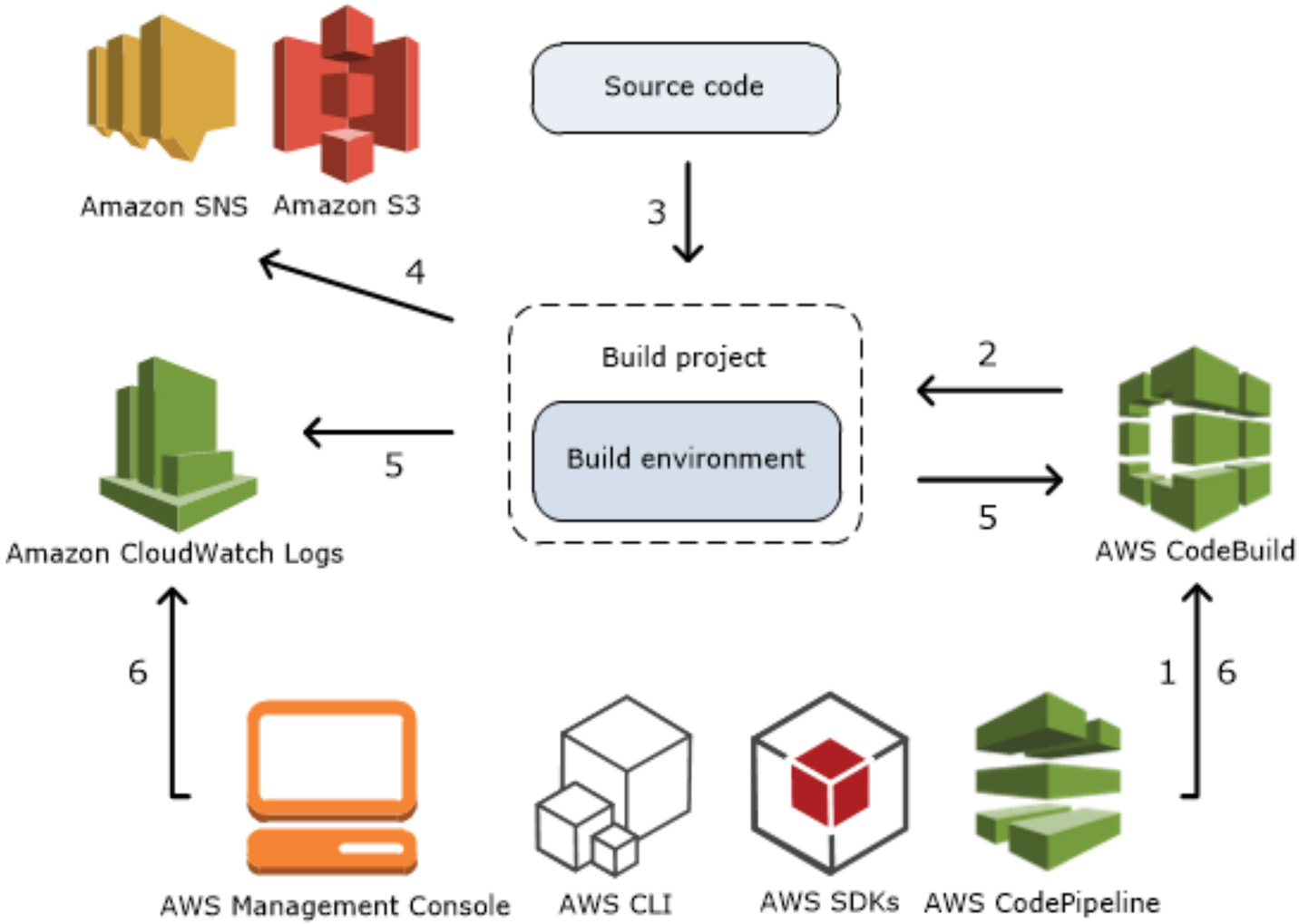
CodeBuild - Introduction

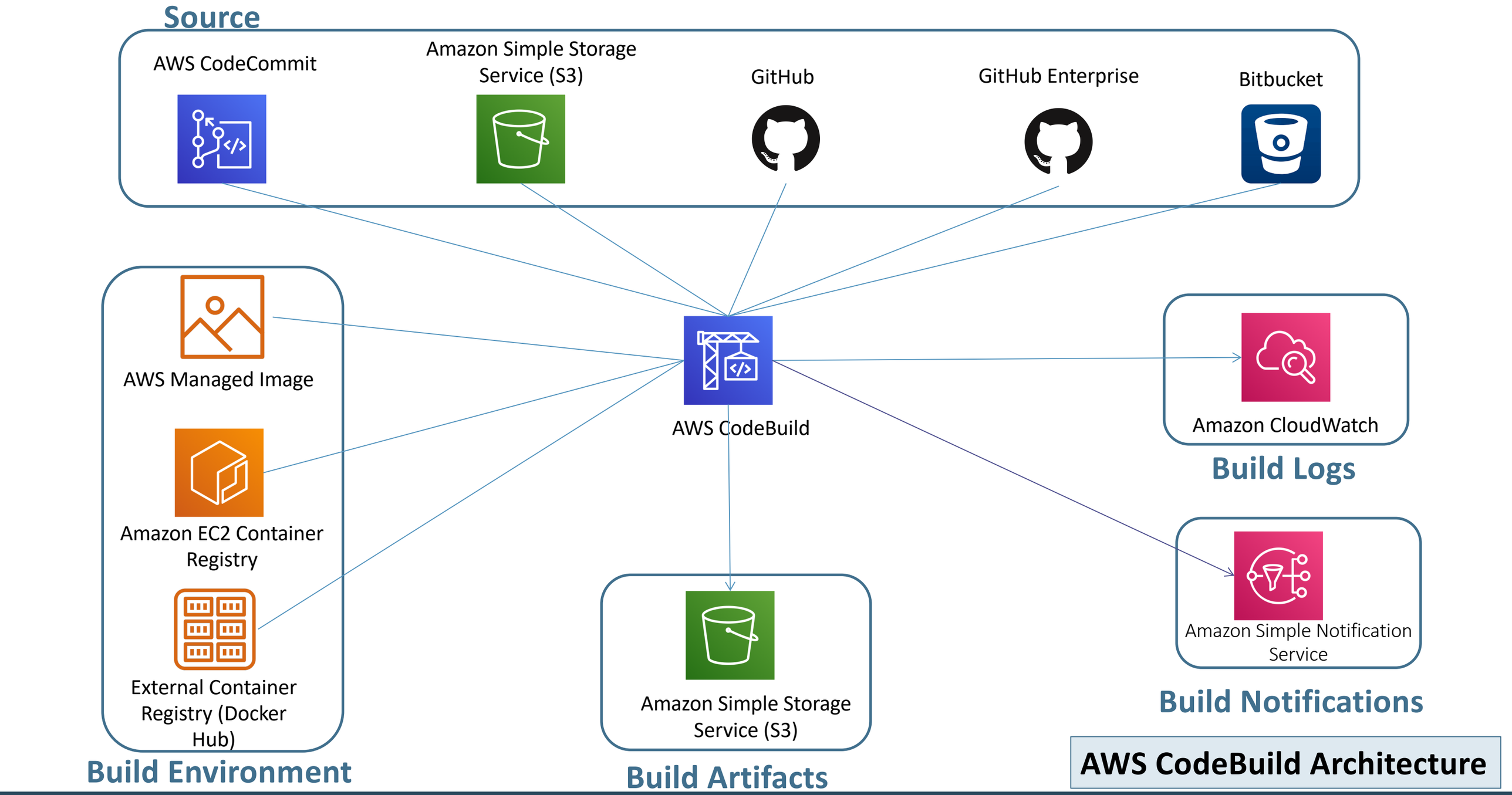
- CodeBuild is a fully managed build service in the cloud.
- Compiles our source code, runs unit tests, and produces artifacts that are ready to deploy.
- Eliminates the need to provision, manage, and scale our own build servers.
- It provides prepackaged build environments for the most popular programming languages and build tools such as Apache Maven, Gradle, and many more.
- We can also customize build environments in CodeBuild to use our own build tools.
- Scales automatically to meet peak build requests.

How to run CodeBuild?



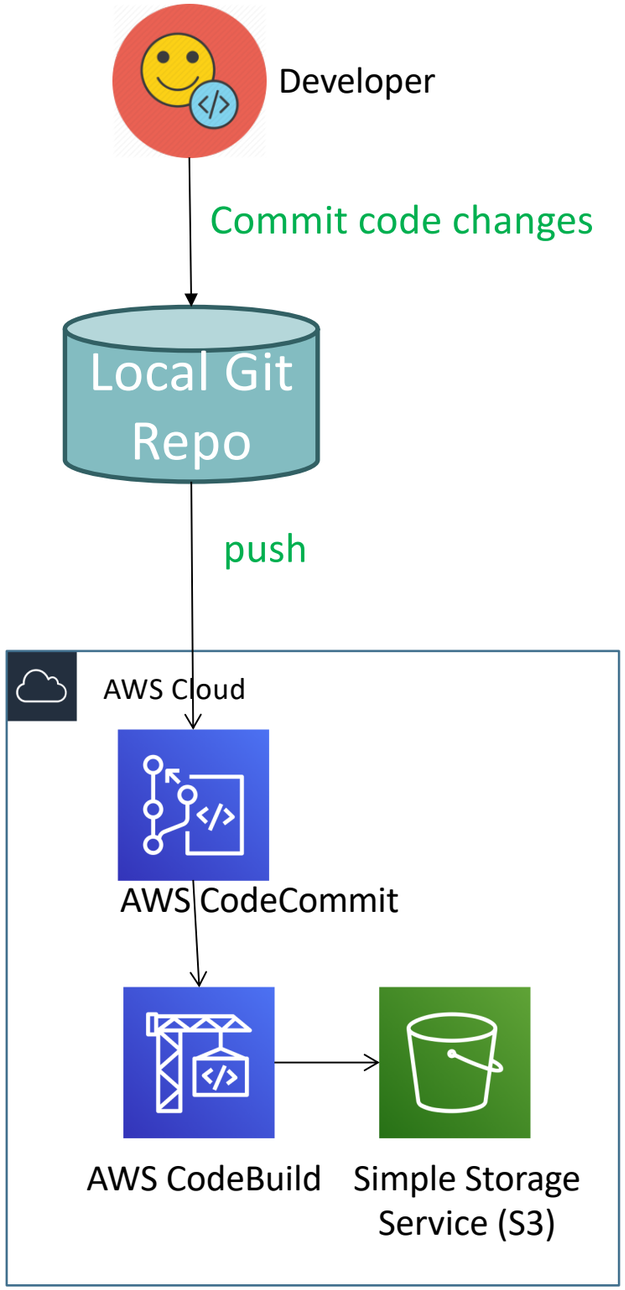
How CodeBuild works?




Source
AWS CodeCommit
Amazon Simple Storage Service (S3)
GitHub
GitHub Enterprise
Bitbucket
AWS Managed Image
Amazon EC2 Container Registry
External Container Registry (Docker Hub)
Build Environment
AWS CodeBuild
Amazon CloudWatch
Build Logs
Amazon Simple Storage Service (S3)
Build Artifacts
Amazon Simple Notification Service
Build Notifications
AWS CodeBuild Architecture

CodeBuild - Steps

- Step#1: Create CodeBuild Project
  - Create a S3 bucket and folder
  - Create CodeBuild project
  - Start build, Verify build logs, Verify build phase details
- Step#2: buildspec.yml & Start Build
  - Create buildspec.yml and check-in code
  - Start build, Verify build logs, Verify build phase details
  - Download the artifacts from S3, unzip and review
  - Run one more build and see versioning in S3.
- Step#3: Create Build Notifications
  - Create state change notification
  - Create Phase change notification



AWS CodePipeline

CodePipeline - Introduction

- AWS CodePipeline is a continuous delivery service to model, visualize, and automate the steps required to release your software.
- Benefits
  - We can automate our release processes.
  - We can establish a consistent release process.
  - We can speed up delivery while improving quality.
  - Supports external tools integration for source, build and deploy.
  - View progress at a glance
  - View pipeline history details.

AWS CodePipeline Architecture

AWS CodePipeline

AWS CodeCommit

Amazon EC2 Container Registry

Simple Storage Service (S3)

GitHub

Source

AWS CodeBuild

Jenkins

Build

AWS CloudFormation

AWS CodeDeploy

AWS Elastic Beanstalk

AWS Service Catalog

Amazon Elastic Container Service

Amazon Elastic Container Service (Blue/Green)

Simple Storage Service (S3)

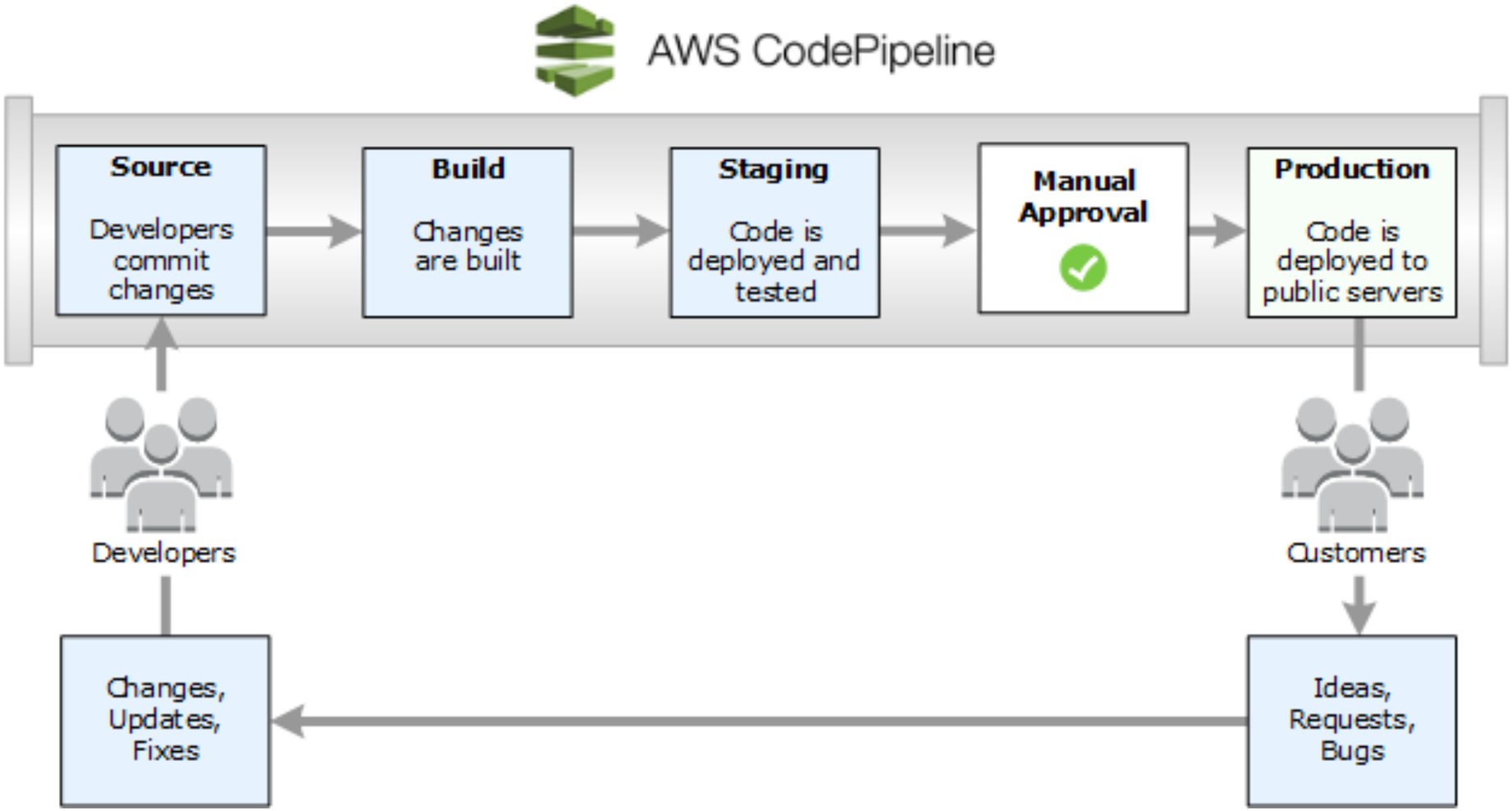
Deploy

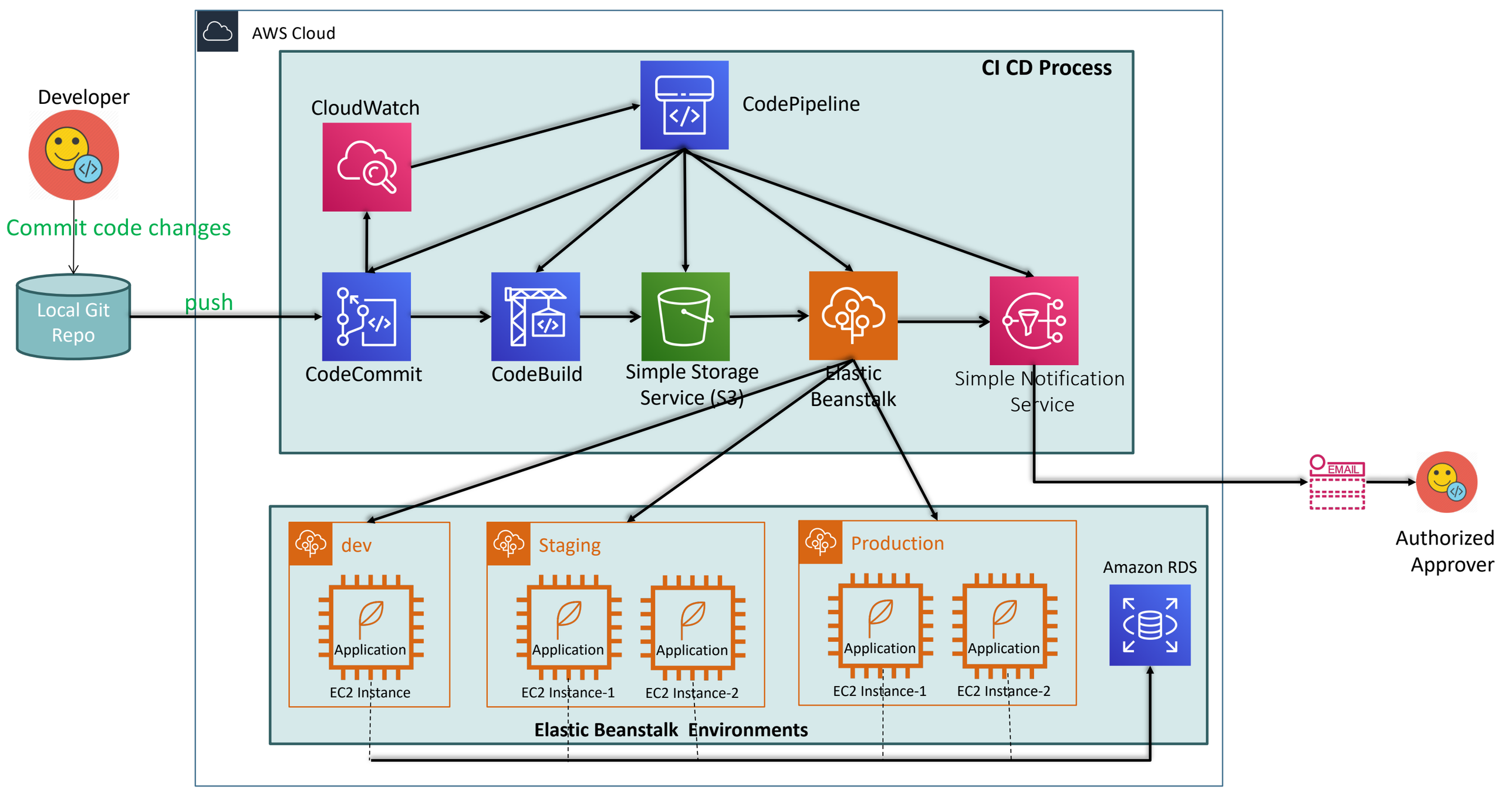
Amazon CloudWatch

GitHub Webhooks

Monitor Source Changes

Continuous Delivery




AWS Cloud
CI CD Process
Developer
Commit code changes
Local Git Repo
push
CloudWatch
CodePipeline
CodeCommit
CodeBuild
Simple Storage Service (S3)
Elastic Beanstalk
Simple Notification Service
EMAIL
Authorized Approver
dev
Application
EC2 Instance
Staging
Application
Application
EC2 Instance-1
EC2 Instance-2
Production
Application
Application
EC2 Instance-1
EC2 Instance-2
Amazon RDS
Elastic Beanstalk Environments

CodePipeline - Steps

- Step-1: Create Pipeline
  - Source: CodeCommit
  - Build: CodeBuild
  - Artifacts: S3
  - Deploy: ElasticBeanstalk – Dev Environment

- Step-2: Make changes to Application & Check-In Code
  - Make changes to rest app and check-in
  - Pipeline should trigger the build automatically.

CodePipeline – Manual Approval & Prod Deployment

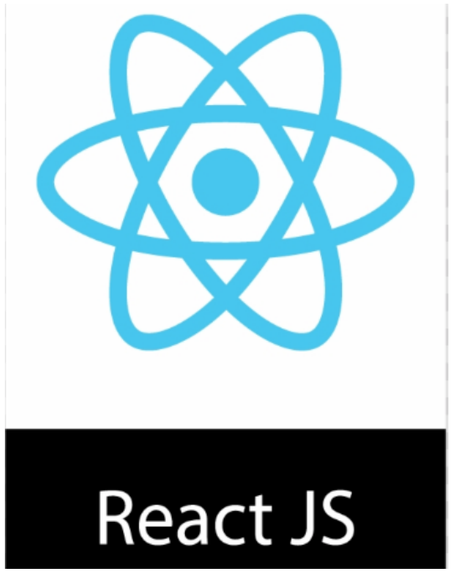
- Step-3: Create Staging Deployment Stage in CodePipeline
- Step-4: Create Manual Approval stage in CodePipeline
- Step-5: Create Prod Deployment stage in CodePipeline .
- Step-6: Check-in changed code to trigger pipeline and monitor the pipeline process.

AWS CloudFront

AWS S3 Static Sites

AWS Elastic Beanstalk

Full Stack Application CI CD Implementation



Spring Boot Restful API

CI CD – For ReactJS Application

- Step-1: Understand ReactJS manual build process and automate it.
  - Install Node Modules: npm install
  - Run locally: npm run start
  - Production Build steps
    - Create production Build: npm run build
    - Upload build folder content to S3
    - Invalidate Cloud Front cache
- Step-2: Setup local and remote git repository
  - Add .gitignore file
  - git init
  - git add .
  - git commit -am “first commit”
  - Create repo in github
  - git remote add origin <repo-url>
  - git push --set-upstream origin master

CI CD – For ReactJS Application

- Step-3: Understand files listed below
  - buildspec.yml (update s3 bucket name)
  - .env.development
  - .env.production (update with devapi31 url)

- Step-4: Create pipeline
  - Source: github
  - Build: CodeBuild
    - Create two custom IAM policies (for s3 bucket, for CloudFront invalidate)
  - Test by updating version value in src/Auth/properties.js

Elastic Beanstalk
EB CLI

EB CLI

- EB CLI is a command line interface for Elastic Beanstalk that provides interactive commands that simplify creating, updating and monitoring environments from a local repository.
- We can use the EB CLI as part of our everyday development and testing cycle as an alternative to the AWS Management Console.
- We need to use EB CLI 3.0 or higher. We are using EB CLI 3.14.6 in this course. Older versions has different set of commands so its good to use the latest.

EB CLI Commands

- eb init
- eb status
- eb events
- eb health
- eb open
- eb list
- eb list –all
- eb terminate
- eb abort

- eb clone
- eb swap
- eb appversion
- eb logs
- eb scale
- eb deploy
- eb deploy –staged
- eb codesource codecommit

EB CLI Steps

- Step-1: Install EB CLI
- Step-2: Setup Development Environment in STS IDE
- Step-3: EB CLI Pre-requisites for a development projects
- Step-4: Create EB Application and EB environment
- Step-5: EB CLI Commands
- Step-6: Deploy updates to application
- Step-7: Create new environment (qa)
- Step-8: EB CLI & CodeCommit Integration
- Step-9: Configure CodeCommit Interactively
- Step-10: Advanced environment Customizations with .ebextensions

EB CLI - Steps

- Step-1: Install EB CLI
  - Option-1: Using brew
    - brew install zlib openssl readline
    - brew install aws-elasticbeanstalk
    - brew list aws-elasticbeanstalk
    - eb --version
    - Reference: https://formulae.brew.sh/formula/aws-elasticbeanstalk
  - Option-2: Using EB CLI Installer
    - brew install zlib openssl readline
    - git clone https://github.com/aws/aws-elastic-beanstalk-cli-setup.git
    - ./aws-elastic-beanstalk-cli-setup/scripts/bundled_installer
    - eb --version
    - Reference: https://github.com/aws/aws-elastic-beanstalk-cli-setup
  - Option-3: Manually Install EB CLI
    - Refer below link for detailed instructions
    - Reference: https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/eb-cli3-install-advanced.html

EB CLI - Steps

- Step-02: Setup Development Environment
  - Create a folder ”ebcli-apps” and copy project located in course uploads “AWS-ElasticBeanstalk-Masterclass-Course-Artifacts/EB-CLI/03-ebcli-h2.zip”
  - Unzip and import the project to STS IDE
  - Run the Application and access sample hello world API.

EB CLI - Steps

- Step-3: EB CLI Pre-requisites for a development projects
  - Create CodeBuild Role
    - CodeCommit – Full Access
    - S3 – Full Access
    - CloudWatch – Full Access
    - Elastic Beanstalk – Full Access
  - Update buildspec.yml
    - Artifacts
    - CodeBuild Settings

```
artifacts:
  files:
    - 'target/03-ebcli-h2.jar'
    - '.ebextensions/*'
    - 'Procfile'
```

```
version: 0.2

eb_codebuild_settings:
  CodeBuildServiceRole: arn:aws:iam::180789647333:role/EBCLI-CodeBuildRole
  ComputeType: BUILD_GENERAL1_SMALL
  Image: aws/codebuild/standard:2.0
  Timeout: 10
```

EB CLI - Steps

- Step-4: Create Application and environment
  - Create Application: eb init
  - Create Environment: eb create
    - Watch codebuild logs in parallel
    - Note: Temporary codebuild projects gets created, generates artifacts, stores in S3 and will get deleted after that.
    - Watch the newly created environment in AWS management console.

EB CLI Steps

- Step-5: EB CLI Commands
  - eb status: Provides information about the status of the environment.
  - eb health: Returns the most recent health for the environment.
  - eb events: Returns the most recent events for the environment.
  - eb open: Opens the public URL of your website in the default browser.
  - eb list: Lists all environments in the current application
  - eb list –all: Lists all environments in all applications
  - eb terminate: Terminates the running environment
  - eb abort: Cancels an upgrade when environment configuration changes to instances are still in progress
  - eb clone: Clones an environment to a new environment so that both have identical environment settings.
  - eb swap: Swaps the environment's CNAME with the CNAME of another environment
  - eb appversion: Manages your Elastic Beanstalk application versions, including deleting a version of the application or creating the application version lifecycle policy

EB CLI Steps

- Step-5: EB CLI Commands
  - eb logs: This command has two distinct purposes: to enable or disable log streaming to CloudWatch Logs, and to retrieve instance logs or CloudWatch Logs.
    - eb logs –instance <instance-id>
    - eb logs –all –zip or --stream
    - eb logs -cloudwatch-logs
      - eb logs –cw
      - https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/eb3-logs.html

EB CLI Steps

- Step-6: Deploy updates to application
  - Access hello world service before updates
  - Update the hello world service
  - Verify app version - eb appversion
  - eb deploy: Deploys the application source bundle from the initialized project directory to the running application
  - Verify hello world service – latest changes should reflect
  - eb status
  - eb scale: Scales the environment to always run on a specified number of instances, setting both the minimum and maximum number of instances to the specified number.
    - eb scale number-of-instances
    - eb scale number-of-instances environment-name

EB CLI Steps

- Step-7: Create new environment (qa)
  - Eb create <environment name>: Not recommended, it creates the environment with classic load balancer (CLB)
  - eb create: Select interactive options after executing eb create so we have an option to choose the load balancer. Always recommended to choose Application Load Balancer (ALB) if we have an option to choose CLB or ALB.
  - eb use: Sets the specified environment as the default environment.

EB CLI Steps

- Step-8: EB CLI & CodeCommit Integration
  - We can use the EB CLI to deploy our applications directly from our AWS CodeCommit repository.
  - With CodeCommit, we can upload only our changes to the repository when we deploy, instead of uploading our entire project during every change or release.
  - EB CLI pushes our local commits and uses them to create application versions when we use **eb create** or **eb deploy**.
  - Some regions don't offer CodeCommit. The integration between Elastic Beanstalk and CodeCommit doesn't work in these regions.

EB CLI Steps

- Step-8: EB CLI & CodeCommit Integration
  - Git commands for creating local git repository
    - git init
    - git add .
    - git commit –am “V1-FirstCommit-CCIntegration”
  - Creating codecommit repository using eb cli
    - eb init
  - Deploying from codecommit repository
    - eb deploy
      - Pushes new local commits to code commit repository
      - CodeBuild uses HEAD revision of branch to create the archive
      - Deploys to EB environment.
    - eb deploy --staged
      - As we develop or debug, we might not want to push changes that we haven't confirmed are working.
      - We can avoid committing our changes by staging them and using **eb deploy --staged** (which performs a standard deployment).

EB CLI steps

- Step-9: Configure CodeCommit Interactively
  - eb codesource codecommit
  - To disable CodeCommit integration
    - eb codesource local

EB CLI Steps

- Step-10: Advanced environment Customizations with .ebextensions
  - We can add Elastic Beanstalk configuration files (.ebextensions) to our web application's source code to configure our environment and customize the AWS resources that it contains
  - Supports YAML or JSON formatted documents with .config file extension.
  - Files need to be placed in .ebextensions folder
  - I recommend using YAML which is more flexible and readable than JSON.
  - Always test new .config files in test environments else if something wrong in these files might create problems for entire environment.
  - Additional References
    - https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/ebextensions.html
    - https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/command-options-general.html#command-options-general-elbhealthcheck
    - https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/ebextensions-optionsettings.html
    - https://docs.aws.amazon.com/codecommit/latest/userguide/troubleshooting-ch.html#troubleshooting-macoshttps
    - https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/java-se-procfile.html
    - https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/command-options.html

EB CLI Steps

- Procfile
  - If we have more than one JAR file in the root of our application source bundle, we must include a Procfile file that tells Elastic Beanstalk which JAR(s) to run.
  - We can also include a Procfile file for a single JAR application to configure the Java virtual machine (JVM) that runs our application.
  - We must save the Procfile in our source bundle root.
  - The file name is case sensitive.

Procfile

```
web: java -jar server.jar -Xms256m
cache: java -jar mycache.jar
web_foo: java -jar other.jar
```

EB CLI Steps

- **Option Settings**
- We can use the option_settings key to modify the Elastic Beanstalk configuration and define variables that can be retrieved from our application using environment variables.
- Some namespaces allow us to extend the number of parameters, and specify the parameter names.

Syntax

The standard syntax for option settings is an array of objects, each having a `namespace`, `option_name` and `value` key.

```
option_settings:
  - namespace:  namespace
    option_name:  option name
    value:  option value
  - namespace:  namespace
    option_name:  option name
    value:  option value
```

The `namespace` key is optional. If you do not specify a namespace, the default used is `aws:elasticbeanstalk:application:environment`:

```
option_settings:
  - option_name:  option name
    value:  option value
  - option_name:  option name
    value:  option value
```

Elastic Beanstalk also supports a shorthand syntax for option settings that lets you specify options as key-value pairs underneath the namespace:

```
option_settings:
  namespace:
    option name: option value
    option name: option value
```

EB CLI Steps

- Step-10: Advanced environment Customizations with .ebextensions
  - Create Procfile & .ebextensions folder in our application root folder
  - Create healthcheckurl.config

Procfile

Running-Notes > Procfile

```
1  web: java -jar target/03-ebcli-h2.jar
```

healthcheckurl.config

Course-Uploads > AWS-ElasticBeanstalk-Masterclass-Course-Artifacts > EB-CLI > healthcheckurl.config

```
1  option_settings:
2    - namespace:  aws:elasticbeanstalk:environment:process:default
3      option_name:  HealthCheckPath
4      value:  /health/status
5    - namespace:  aws:elasticbeanstalk:environment:process:default
6      option_name:  MatcherHTTPCode
7      value:  200
```

Elastic Beanstalk

What is Packer?

- Packer is an open source tool for creating identical machine images for multiple platforms from a single source configuration.
- Packer is lightweight, runs on every major operating system, and is highly performant, creating machine images for multiple platforms in parallel.
- Packer does not replace configuration management tools like Chef or Puppet.
- In fact, when building images, Packer is able to use tools like Chef or Puppet to install software onto the image.
- A *machine image* is a single static unit that contains a pre-configured operating system and installed software which is used to quickly create new running machines.
- Machine image formats change for each platform. Some examples include AMIs for EC2, VMDK/VMX files for VMware, OVF exports for VirtualBox, etc.

Why Packer for Elastic Beanstalk?

- We will be using Packer to create EC2 AMI's that can be used by Elastic Beanstalk for creating custom platforms.

Packer - Steps

- Step-1: Install Packer
  - brew install packer
  - Verify installation
    - packer version
    - packer
  - Reference: https://www.packer.io/intro/getting-started/install.html
- Step-2: Packer Template Fundamentals
  - Variables
  - Builders
  - Provisioners
  - Communicators

Packer - Steps

- User Variables
  - User variables allow our templates to be further configured with variables from the command-line, environment variables, Vault, or files.
  - This lets us parameterize our templates so that we can keep secret tokens, environment-specific data, and other types of information out of our templates.
  - This maximizes the portability of the template.

Packer - Steps

- Builders
  - Builders are responsible for creating machines and generating images from them for various platforms.
  - For example, there are separate builders for EC2, VMware, VirtualBox, etc.
  - Packer comes with many builders by default, and can also be extended to add new builders.
  - Every build is associated with a single communicator.
- Communicators
  - Communicators are used to establish a connection for provisioning a remote machine (such as an AWS instance).
  - Communicators are the mechanism Packer uses to upload files, execute scripts, etc. with the machine being created.
- Reference
  - https://www.packer.io/docs/builders/amazon.html

```
"builders": [
  {
    "type": "amazon-ebs",
    "profile": "default",
    "region": "{{user `region`}}",
    "instance_type": "t2.micro",
    "source_ami": "ami-0b69ea66ff7391e80",
    "ssh_username": "ec2-user",
    "ami_name": "docker-17.12.1-ce",
    "ami_description": "Amazon Linux Image with Docker-CE",
    "run_tags": {
      "Name": "packer-builder-docker",
      "Tool": "Packer",
      "Author": "kalyan"
    }
  }
],
```

Packer - Steps

- Provisioners
  - Provisioners section contains an array of all the provisioners that Packer should use to install and configure software within running machines prior to turning them into machine images.
  - In simple terms, install desired software on machine before turning them as images.
  - Provisioners are *optional*.
  - If no provisioners are defined within a template, then no software other than the defaults will be installed within the resulting machine images.
  - A provisioner definition is a JSON object that must contain at least the type key. This key specifies the name of the provisioner to use.

```
"provisioners": [
  {
    "type": "shell",
    "script": "./setup.sh"
  }
]
```

setup.sh

```
#/bin/sh

sudo yum update -y
sudo yum install docker -y
sudo service docker start
sudo usermod -aG docker ec2-user
```

Packer - Steps

- Step-3: Understand packer project
  - Download packer project from course-uploads
    - AWS-ElasticBeanstalk-Masterclass-Course-Artifacts/EB-CustomPlatforms/packer1.zip
  - Review and understand below two files
    - ami.json
    - setup.sh
- Step-4: Execute packer commands to create EC2 AMI
  - packer build <template file>
  - packer build ami.json
  - Understand the output from packer build command.
  - Navigate to AWS management console EC2 → Instances, verify packer-builder-docker vm got created.
  - Navigate to AWS management console Images → AMIs, verify docker ami getting created.

Packer - Steps

- Step-5: Create EC2 Instance from new AMI created.
  - Create EC2 Instance from new AMI created
  - Login to VM and verify the packages.
    - rpm -qa | grep docker
    - docker images
    - docker run hello-world
    - docker images

Elastic Beanstalk
Custom Platforms

Elastic Beanstalk - Custom Platforms

- A custom platform is a more advanced customization than a custom image in several ways.
- A custom platform lets us develop an entire new platform from scratch, customizing the operating system, additional software, and scripts that Elastic Beanstalk runs on platform instances.
- This flexibility enables us to build a platform for an application that uses a language or other infrastructure software, for which Elastic Beanstalk doesn't provide a managed platform.
- In addition, with custom platforms we use an automated, scripted way to create and maintain our customization, whereas with custom images we make the changes manually over a running instance.

Elastic Beanstalk - Custom Platforms

- To create a custom platform, we build an AMI from one of the supported operating systems—Ubuntu, RHEL, or Amazon Linux.
- We create our own Elastic Beanstalk platform using **Packer**, which is an open-source tool for creating machine images for many platforms, including AMIs for use with Amazon Elastic Compute Cloud (Amazon EC2).
- An Elastic Beanstalk platform comprises an
  - AMI configured to run a set of software that supports an application
  - metadata that can include custom configuration options and default configuration option settings.

Elastic Beanstalk - Custom Platforms

- Elastic Beanstalk manages Packer as a separate built-in platform, and we don't need to worry about Packer configuration and versions.
- We create a platform by providing Elastic Beanstalk with a Packer template, and the scripts and files that the template invokes to build an AMI.
- These components are packaged with a platform definition file, which specifies the template and metadata, into a ZIP archive, known as a platform definition archive.
- When we create a custom platform, we launch a single instance environment without an Elastic IP that runs Packer.
- Packer then launches another instance to build an image. We can reuse this environment for multiple platforms and multiple versions of each platform.

Elastic Beanstalk - Custom Platforms

- Custom platforms are AWS Region specific. If we use Elastic Beanstalk in multiple Regions, we must create our platforms separately in each Region.

Elastic Beanstalk - Custom Platforms Steps

- Step-1: Download and Unzip from course artifacts
  - Create folder EB-CustomPlatforms
  - Download custom platform project from course-uploads
    - AWS-ElasticBeanstalk-Masterclass-Course-Artifacts/EB-CustomPlatforms/custom_platform.zip
  - Unzip the project in EB-CustomPlatforms folder
- Step-2: Understand the following on a high level
  - Packer Template: tomcat_platform.json.
  - Custom Platform file: platform.yml
  - builder.sh, CONFIG
  - Folder: setup-scripts
  - Folder: platform-uploads
  - References:
    - https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/custom-platforms.html
    - https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/platform-yaml-format.html

Elastic Beanstalk - Custom Platforms Steps

- Step-3: Create Custom Platform
  - Initialize a platform repository
    - eb platform init
  - Create Platform – Builds a new version of platform (custom-tomcat)
    - eb platform create
  - Verify Status
    - eb platform status
  - Verify platform logs
    - eb platform logs
  - Verify platform events
    - eb platform events
  - List the version of current platform
    - eb platform list
- EB Platform Command Reference: https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/eb3-platform.html

Elastic Beanstalk - Custom Platforms Steps

- Step-4: Create Elastic Beanstalk environment using newly created custom platform
  - Create EB Environment
    - Create Application: EB-Custom-Platform-Demo
    - Application Versions Page: Upload demo.war file to Application Versions page.
    - Create EB Environment: demoapp1
    - Select Custom Platform: custom-tomcat
- Step-5: Publish new version of Custom Platform
  - Update platform.yml with new environment Variable
  - Create new version of platform
    - eb platform create
  - Important Note: Refresh EB environments page on AWS management console.
  - Apply new platform changes to demoapp1 environment.

Elastic Beanstalk - Custom Platforms Steps

- Step-6: Clean up resources
  - Terminate demoapp1 environment
  - Clean-up custom platform resources from AWS EB.
    - eb platform list
      - List the version of the current platform.
    - eb platform delete
      - Delete a platform version. The version isn't deleted if an environment is using that version.
  - Verify EC2 → AMIs
  - Verify EB → Custom Platforms drop down.
  - Terminate EB Environment named Custom Platform Builder

Thank You