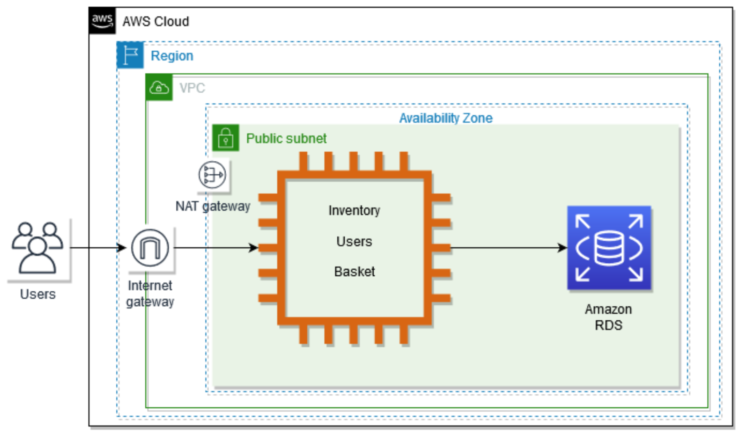
App Modernization with Serverless and Containers

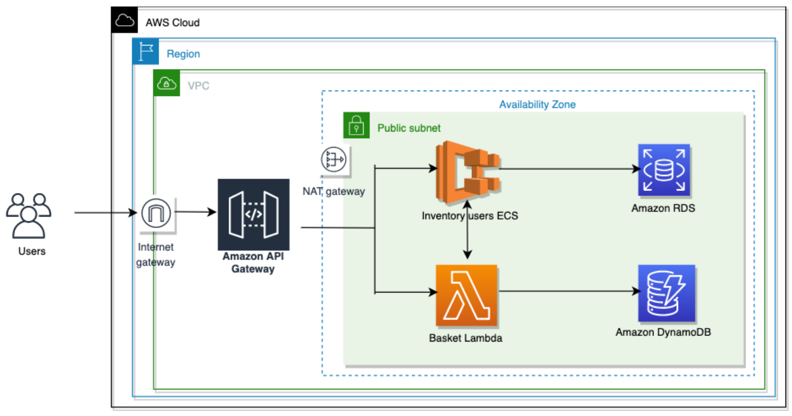
Workshop overview

The workshop walks through the steps of modernizing a sample monolith application leveraging Serverless and Container technologies. There are two parts to the application: a static web portion and a monolith backend application running on an EC2 instance with dependence to MySQL database. We would first introduce API Gateway between the frontend and backend layer as we start replacing some of the functionality in the backend with new set of microservices leveraging Lambda and DynamoDB. We would containerize the application and deploy it on ECS, exposing rest of the functionality through API Gateway. We have adopted the Strangler Fig pattern for modernizing the application with microservices, leveraging Serverless and Containers.

Original architecture



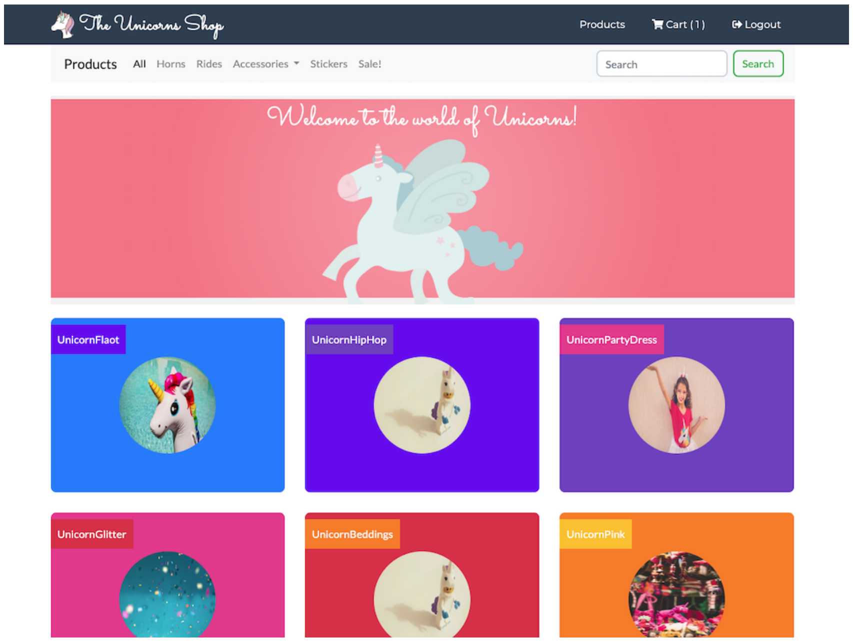
Architecture after App Modernization



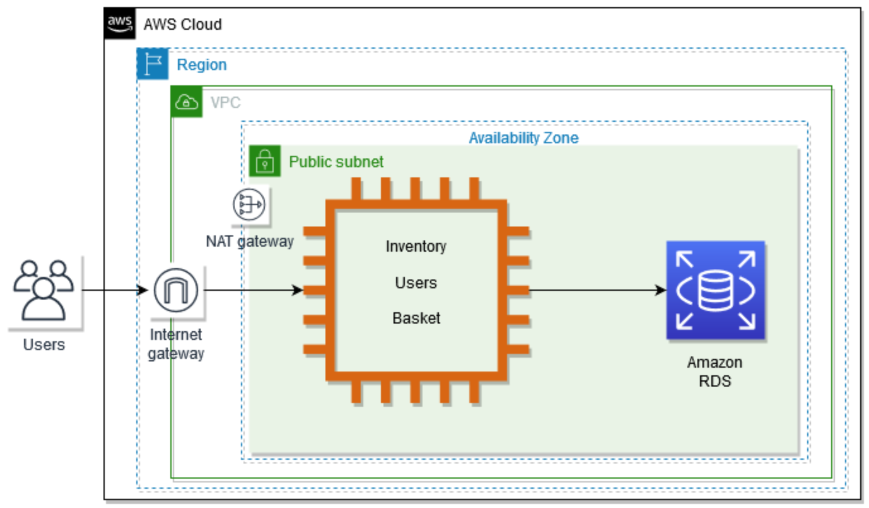
We've broken the workshop down into easy to follow and digestible chunks, which walks you through the process of transforming a monolithic application to a microservices-based application.

In **Module 1**, we will cover the monolithic application. It is a **traditional** Spring Boot Java application which will be deployed on an EC2 instance and connect to RDS MySQL database. The frontend will be hosted on S3 **Static web hosting**, it is a simple yet powerful hosting solution which auto-scale and meet growing needs automatically. Once deployed, the **Unicorns Shop** will be accessible to the outside world. The static frontend would be connecting to the backend application running on EC2 and using a RDS instance to store and retrieve users, baskets.

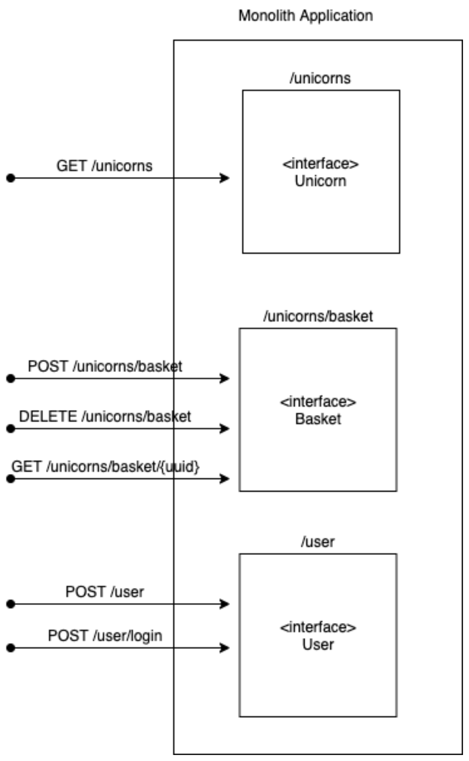
The entire application stack will be deployed using AWS CloudFormation with a separate VPC hosting the EC2 running the spring boot application and talking to a RDS database with the frontend code served from a S3 bucket. The Module 1 would guide the user with the environment creation and setup, connecting the Frontend with the new backend (as the EC2 instance and VPC are newly generated and are available at different IP address). **Note:** If the workshop is hosted by AWS or AWS Partner, the application stack would be prebuilt and user can dive into the "Verify Backend" chapter in Module 1.



Unishop Frontend



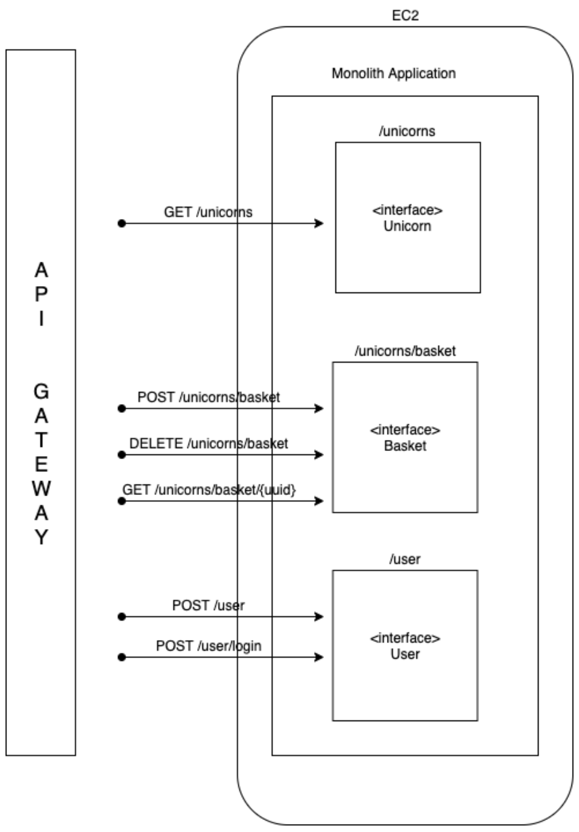
Initial Monolith Architecture of Spring boot application running on EC2



Interfaces and APIs exposed in the Monolith application

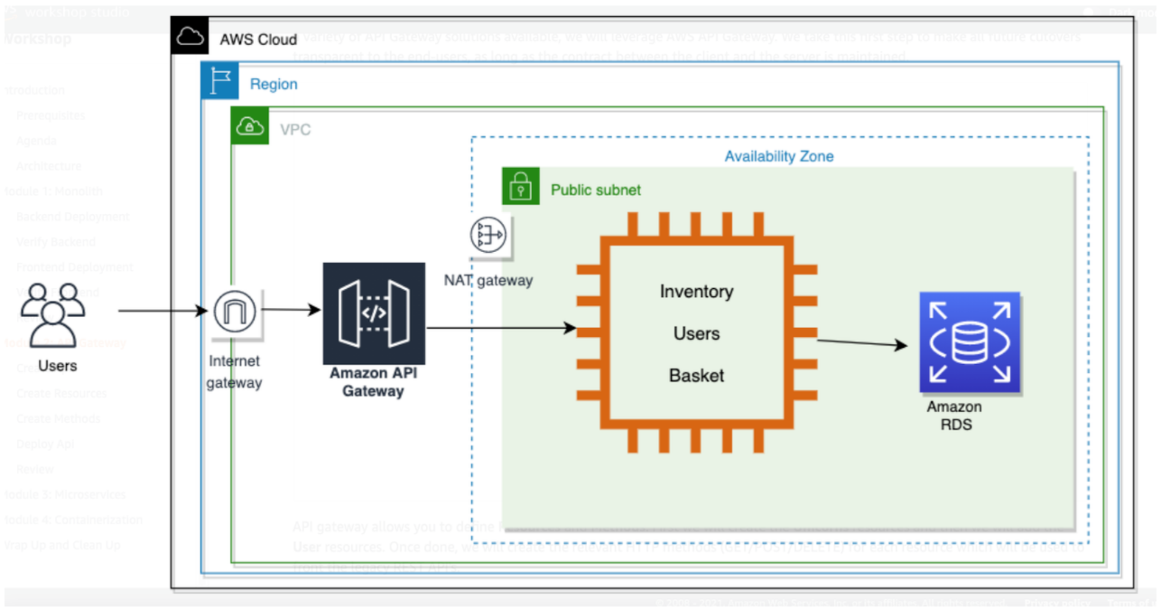
One of the major benefits of moving to microservices architecture is that you can develop each microservice using different technologies stack which is most suitable for the use case. In this case, we decided to use Lambda and DynamoDB as the compute and database capabilities for the Unishop shopping cart functionality. We would use ECS to run a containerized version of the original monolithic application. All these services would be accessed through API Gateway allowing us to slowly replace the original backend monolithic layer with set of microservices using the Strangler Fig Pattern.

In **Module 2-3**, we will extract domain-based functionality and build it as a standalone microservice using Lambda and DynamoDB. In this case, that will be the Unicorn shopping cart functionality. Following are the various interfaces exposed in the backend application for the user, unicorn and basket functionality which will be mediated to go through the API Gateway.

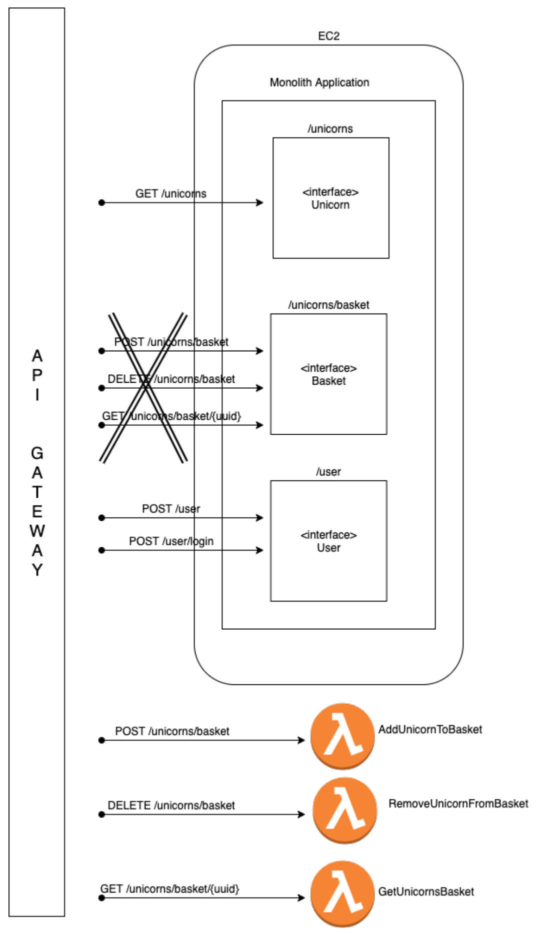


API Gateway integration with Backend running on EC2

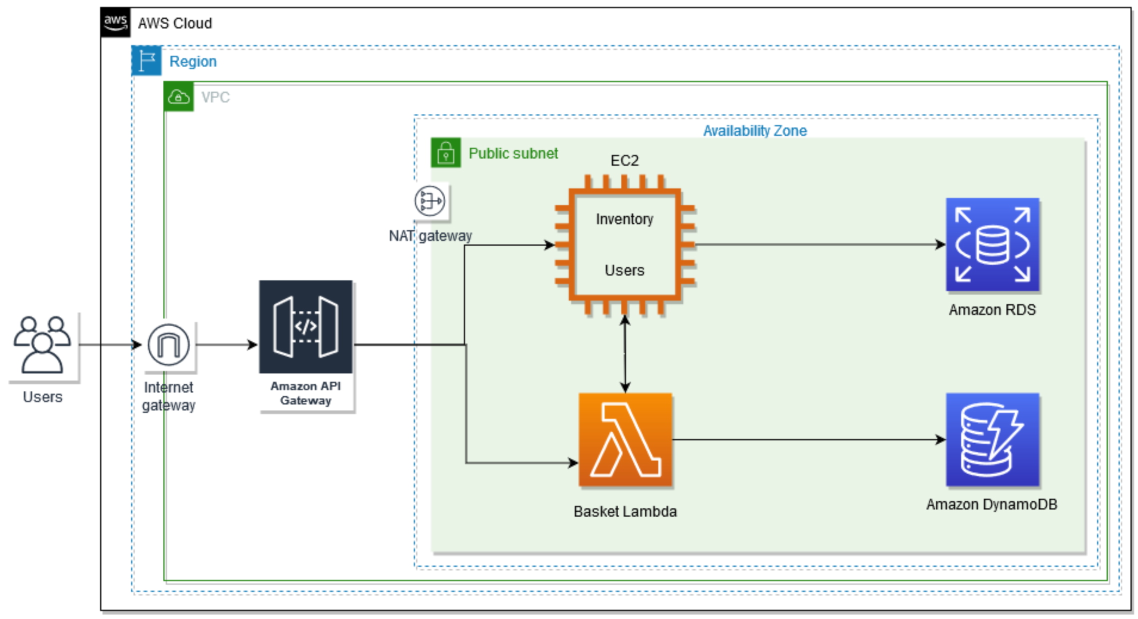
In **Module 2**, we will front the legacy application with API Gateway which will help with switch from old REST API to new Lambda code that handles the unicorn shopping cart. We will create a new REST api with different resources and methods to access the unicorns, basket and user functionality pointing to the backend application.



In **Module 3**, we will be building microservices. We will set up a new DynamoDB table which will hold the shopping cart, followed by deployment of Lambda code which will replace the legacy shopping cart functionality. The API gateway apis would be modified to use the Lambda services for the shopping cart functionality rather than the monolith application hosted on EC2.

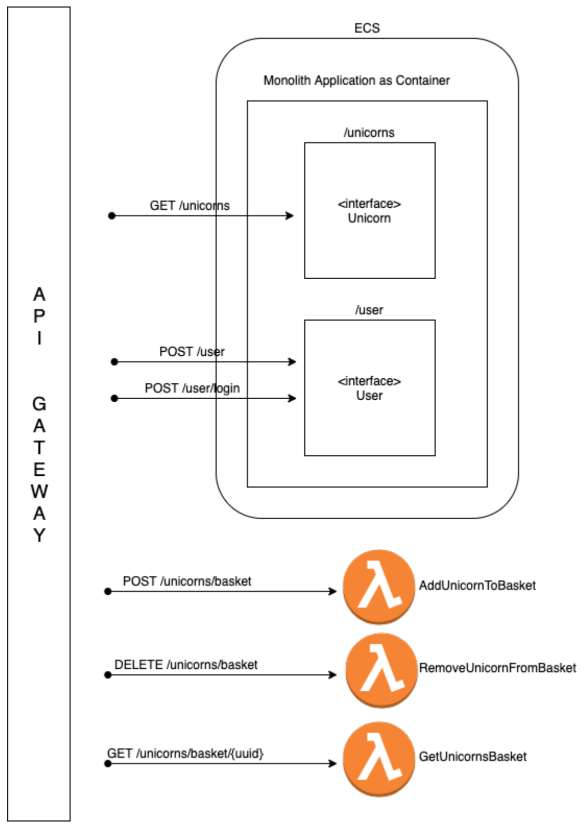


Rerouting api calls to Lambda for Basket operations



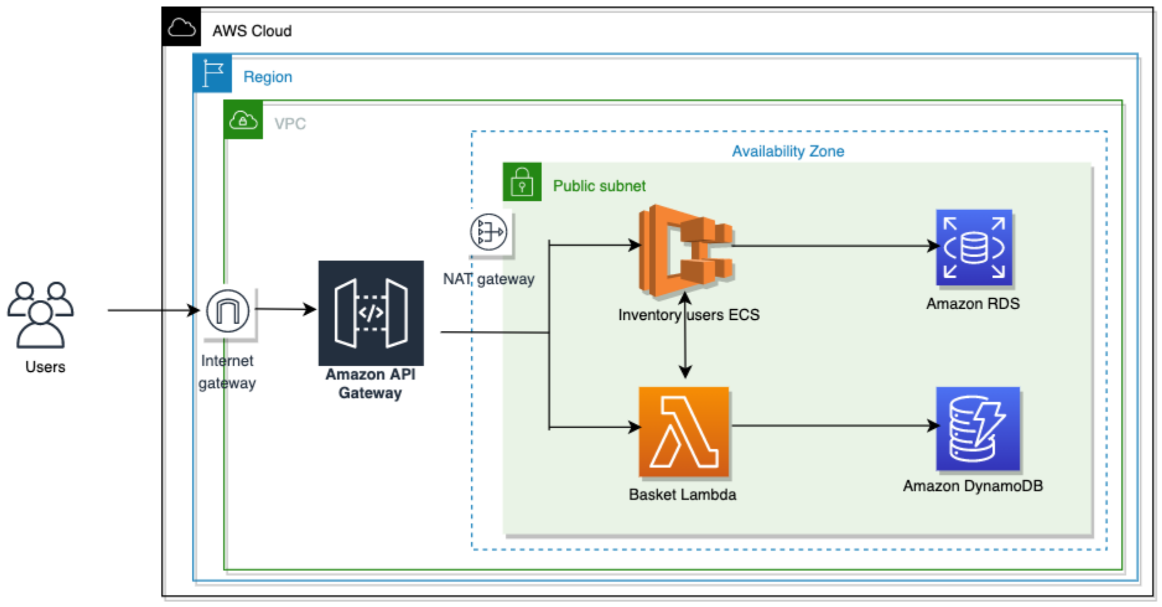
Microservices Architecture with API Gateway + Lambda + DynamoDB

In **Module 4**, we will containerize the existing monolithic application, use ECS to run the containerized workload and expose it via API Gateway.



API Gateway integration with containerized Backend running on ECS

This module demonstrates creation of a dockerized container from an existing application package, deploying and running the container on ECS and exposing it to frontend and external users via API Gateway.



End Architecture with Serverless and Containers

Module 5 will be used to wrap and clean up.

If the user is using a prebuilt AWS environment via AWS or AWS Partner managed event, CloudFormation templates would be executed ahead of time to handle the environment creation, including the initial VPC and application modules deployments; the final cleanup would be also automatically taken care off. If it is managed by the user, one would have to deploy the CloudFormation script and also be responsible for the final environment tear down and clean up.