José Angel Bustamante

Challenge for Tekton

System Design Document

Version 1.0

03/28/2024

Goals

Tekton Challenge using .NET 7 Web API. This project implements Multitenancy, CQRS, Clean Architecture (Onion/Ports & Adapters), Clean Coding standards, Cloud Deployments with Terraform to AWS, Docker Concepts, CICD Pipelines & Workflows and so on.

Features

- Built on .NET 7.0
- Follows Clean Architecture Principles
- Domain Driven Design
- Cloud Ready. Can be deployed to AWS Infrastructure as ECS Containers using Terraform
- Docker-Compose File Examples
- Multi Tenancy Support to allow different test scenarios and strategies:
    - Create Tenants with Multi Database / Shared Database Support
    - Activate / Deactivate Tenants on Demand
- Supports MySQL, MSSQL, Oracle & PostgreSQL
- Security and JWT validation

The following tools and implementations are related to specific design patterns:

- Uses Entity Framework Core as DB Abstraction
- Flexible Repository Pattern
- Dapper Integration for Optimal Performance
- Serilog Integration with various Sinks - File, SEQ, Kibana
- OpenAPI - Supports Client Service Generation
- API Versioning
- Response Caching - Distributed Caching + REDIS
- Fluent Validations
- Audit Logging
- Advanced User & Role Based Permission Management
- Code Analysis & StyleCop Integration with Rulesets
- File Storage Service
- Test Projects
- JWT & Azure AD Authentication
- MediatR - CQRS
- SignalR Notifications

The application implements an asynchronous model following the event-driven design pattern, leveraging SignalR notifications and MediatR.

Prerequisites

Before cloning the repo and creating the solution the following prerequisites must be considered:

- .NET 7 You can find the download here.

Changelogs

View Complete Changelogs.

Assumptions & Capacity Planning

Estimated Numbers:

- 3B page/products views per month
- 30 M updates/deletes per month (due to purchase or administration)
- Read:Write Ratio: 100:1

Then, the “Click-Through rate” is: 1%

- TPS (Transactions Per Second):
- 3B reads per month ->
- 3B / 30 ->
- 3000M / 30
- 1000M / 10
- 100M Reads Per Day
- 100M / 100K ->
- 1M / 1K ->
- 1000 Reads Per Second

How many Writes Per Second?

- 30M per month
- 30M / 30 ->
- 1M Writes Per Day

1M / 100K -> 10 TPS (Transactions Per Second)

Capacity Planning and Storage Estimates:

Catalog (products) information:

- record size: 1KB (worst case)
- includes ProductId, Name, Status(0 o 1), Stock, Description y Price and Image URL
- consider adding 100 products per day to the catalog
- (1KB x 100 x 365) -> 36.5 MB Per Year
- Approximately 5 GB Storage for 5 years

For these cases, a Medium Size Storage (DBMS) and 2 server instances (ECS containers) are sufficient.

Cloud Infrastructure Overview

Cloud Provider: Amazon Web Services (AWS)

Provisioning:

- Terraform: Infrastructure as Code (IaC) is implemented using Terraform. The Terraform configuration files are located in the `Terraform` folder at the project root. Terraform automates the provisioning and management of AWS resources, ensuring consistency and repeatability in infrastructure deployments.

Compute:

- ECS (Elastic Container Service): Docker containers are deployed to ECS for managing and orchestrating containerized applications. ECS provides scalability and flexibility in managing containerized workloads.

Networking:

- ALB (Application Load Balancer): An ALB is used to distribute incoming traffic across multiple ECS container instances. It provides high availability, fault tolerance, and seamless scaling for the application.

Region and Availability Zones:

- Region: us-east-1
    - Availability Zone 1: us-east-1a
    - Availability Zone 2: us-east-1b
- Deploying resources across multiple availability zones ensures redundancy and fault tolerance, enhancing the resilience of the infrastructure.

Virtual Private Cloud (VPC):

- Two VPCs (Virtual Private Clouds): One VPC is created in each availability zone to isolate and segment the network traffic. VPCs provide a secure and private environment for deploying resources.

Database:

- RDS (Relational Database Service): PostgreSQL is used as the default database engine for storing and managing data. RDS offers scalability, high availability, and automated backups, ensuring the reliability and performance of the database.

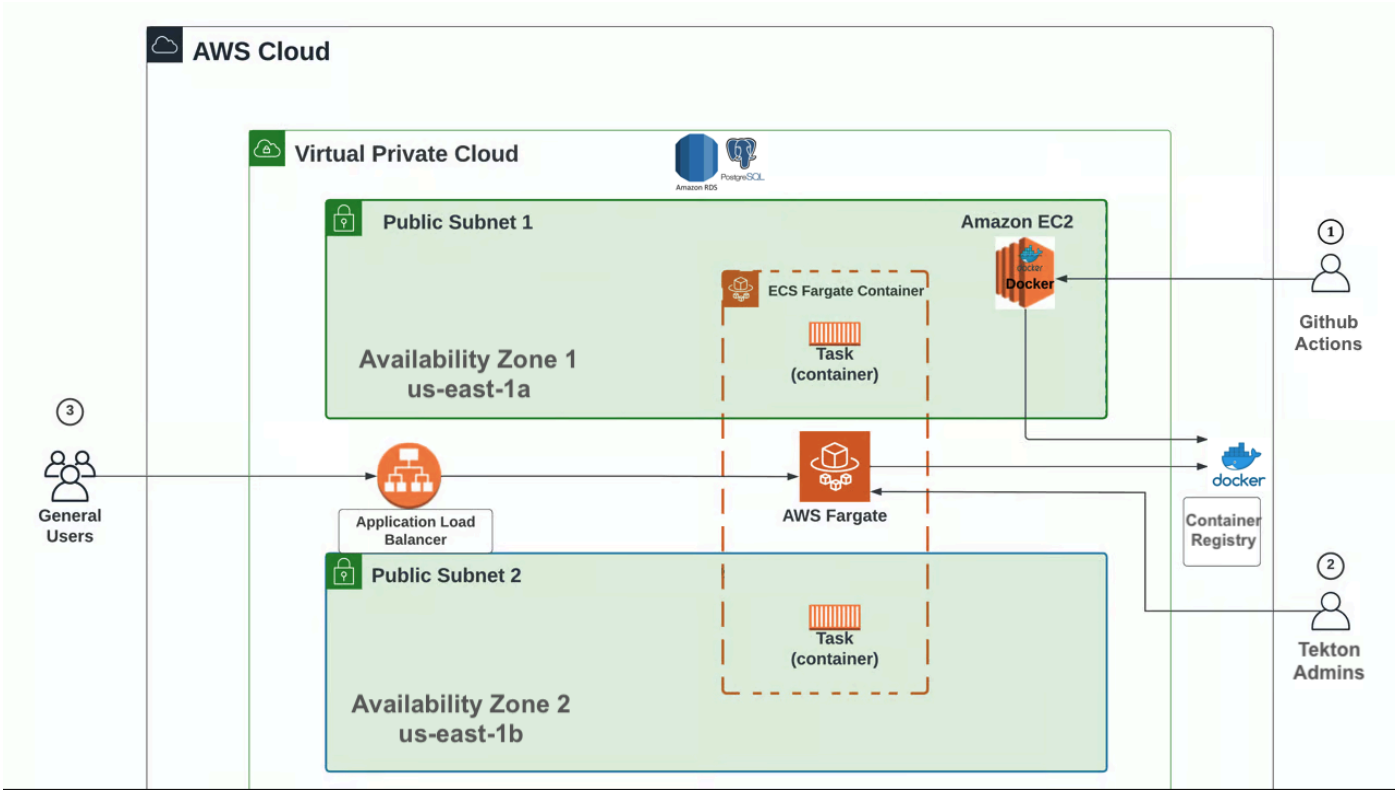
Container Policies with AWS Fargate:

This project utilizes AWS Fargate to manage and run the containerized application without the need to provision and manage servers or clusters.

Scalability Capabilities:

- ECS Auto Scaling: ECS Auto Scaling automatically adjusts the number of ECS tasks based on predefined metrics such as CPU utilization or request count, ensuring optimal resource utilization and accommodating changes in traffic demand.
- ALB Auto Scaling: ALB Auto Scaling dynamically scales the load balancer's capacity to handle varying levels of incoming traffic. It automatically adds or removes ALB instances based on traffic patterns, ensuring consistent performance and availability.
- RDS Scalability: RDS provides scalability options such as Read Replicas and Multi-AZ deployments. Read Replicas allow for scaling read-heavy workloads, while Multi-AZ deployments provide failover support and increased availability.

Infrastructure Diagram



Observability and Monitoring

This is an outline of the monitoring strategy for Tekton WebApi:

ElasticSearch and Kibana Integration:

This project utilizes Environment Variables, along with configuration files, to configure the URL of the Elasticsearch server, enabling seamless integration with Kibana for log aggregation, analysis, and visualization. The .NET7 application uses Serilog to channel the logs to ElasticSearch. Elasticsearch acts as a centralized logging solution, where application logs, metrics, and other relevant data are indexed and stored for analysis. Kibana provides a user-friendly interface for querying and visualizing log data, enabling real-time monitoring, troubleshooting, and trend analysis.

AWS CloudWatch Integration:

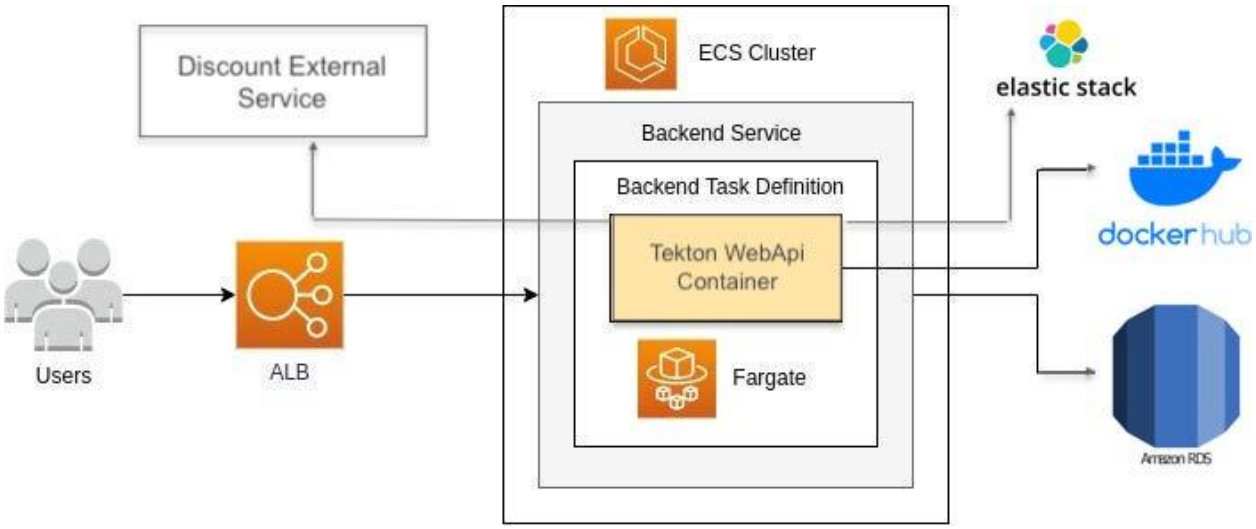
We leverage AWS CloudWatch to monitor the health, performance, and resource utilization of AWS resources and services used in the application deployment. Configure CloudWatch Alarms to automatically trigger notifications or take remedial actions based on predefined thresholds for metrics such as CPU utilization, memory usage, network

traffic, and application-specific metrics. CloudWatch collects the Logs to monitor, and analyze application logs generated by AWS services and resources.

Custom Application Metrics (instrumentation):

Tekton WebApi does not instrument the application code. That's an Anti-Pattern.

Architecture Overview



External Interfaces:

Tekton WebApi interacts with an external service provider in order to get the Discounts per product.

In this case, the system uses mockapi.io with a public profile:

- https://mockapi.io/projects/6603b4cb2393662c31cf74e9

However, the service URL is configurable and can be set as an Environment Variable to be consumed by the containerized application:

- ElasticSearchUrl=http://localhost:9200/
- ExternalDiscountsUrl=https://6603b4cb2393662c31cf74e8.mockapi.io/

This is explained in detail in the Instantiation & Instructions Document.

Record of Changes

Table 1 - Record of Changes

Version Number	Date	Author/Owner	Description of Change
1.0	*03/28/2024*	*Angel Bustamante*	*Initial Version*