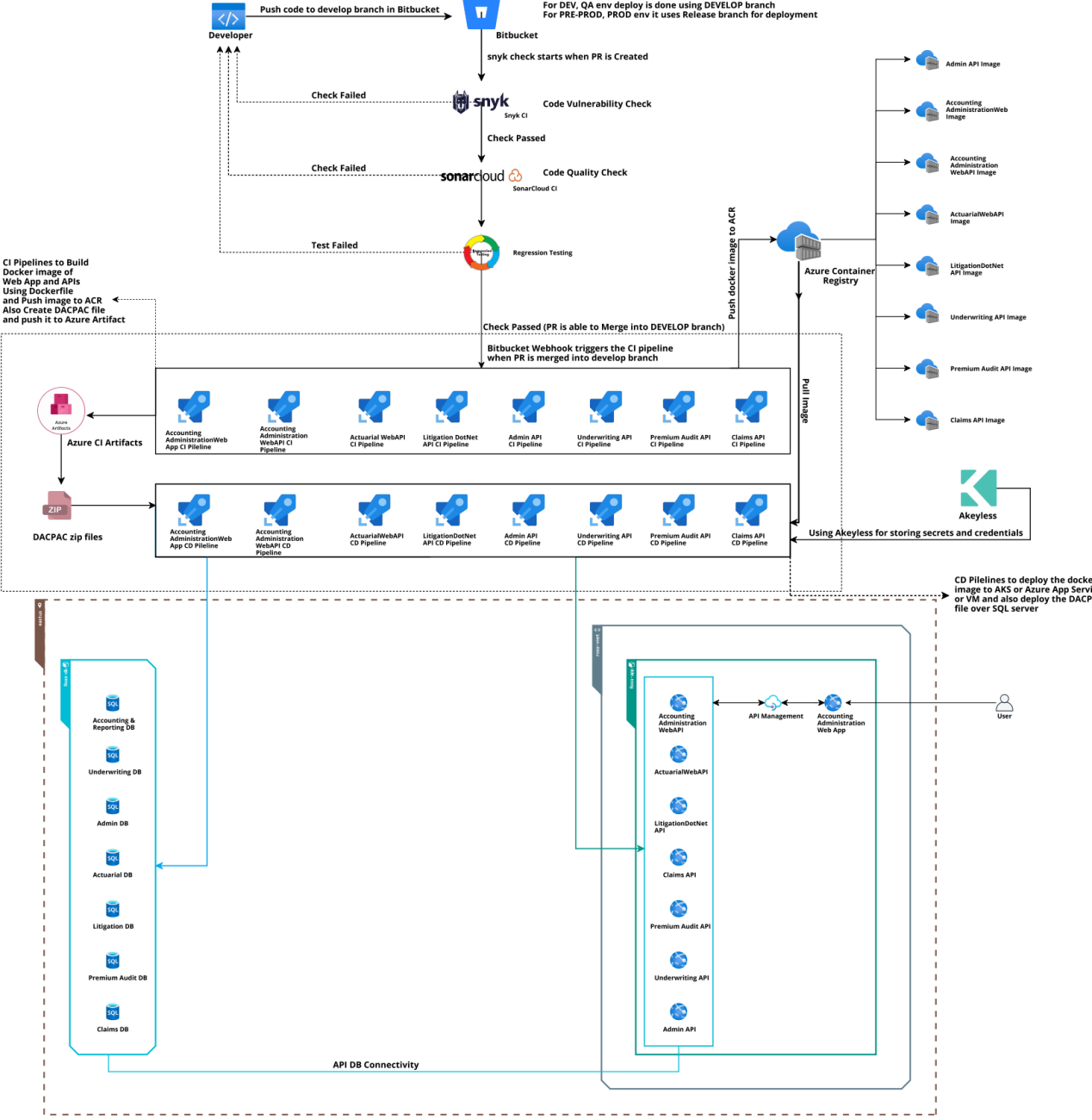
**ROSS APPLICATION WORKFLOW**

1. **WORKFLOW DIAGRAM :-**



1. **INTRODUCTION**

This document outlines the CI/CD workflow for building, testing, and deploying APIs and web applications. The process includes creating Docker images and DACPAC files, using Dockerfiles, and pushing them to Azure Container Registry and Azure Artifact. Deployment pipelines then deploy these images to Azure App Service and manage SQL Server DACPAC deployments. Integrated tools like SonarCloud and Snyk ensure code quality and security by performing checks during pull request merges. Akeyless manages credential storage, and different branches are used for environment-specific deployments to streamline and secure the entire development lifecycle.

1. **Detailed Workflow Description :-**

**3.1. Code Development and Repository Management**

* **Developer:** Developers write and push code to the Bitbucket repository.
* **Bitbucket:** Serves as the central repository for code storage and version control.

**3.2. Continuous Integration (CI) Pipeline**

* **Code Vulnerability Check:**

**Tool:** Snyk CI

**Purpose:** Scans the code for security vulnerabilities.

* **Code Quality Check:**

**Tool:** SonarCloud CI

**Purpose:** Analyzes the code for quality issues.

* **Regression Testing :**

**Triggering Tests**: Regression tests are automatically initiated when a pull

request (PR) is created or merged into the develop branch,

ensuring continuous validation of code changes.

**Ensuring Stability**: These tests verify that recent updates do not disrupt existing

functionality, maintaining application stability throughout

development.

**Integration with CI:** The regression tests are integrated into the CI pipeline,

allowing for immediate feedback and quick identification of

issues before deployment to further environments.

* **Code Merge:**

**Checks failed:** PR (Pull Request) is not allowed to merge.

**Checks passed:** PR can merge into the DEVELOP branch.

* **CI Pipelines:**

**Builds:** Docker images of the web application and APIs.

**Creates:** DACPAC files for database deployments.

**Pushes:** Images to Azure Container Registry (ACR) and DACPAC files to Azure

Artifacts.

**3.3. Continuous Deployment (CD) Pipeline**

* **Azure Container Registry (ACR):** Stores the Docker images.
* **Deployment:**

**Targets:** Docker images are deployed to Azure App Service.

**Database Deployment:** DACPAC files are deployed over SQL servers.

**3.4 Database Management**

* **SQL Server:** Acts as the central database server.
* **Databases:**

Accounting & Reporting DB

Actuarial DB

Admin DB

Claims DB

Litigation DB

Premium Audit DB

Underwriting DB

**3.5. Secrets and Credentials Management**

* **Akeyless:**

**Description:** Akeyless is a secrets management service that securely stores and

manages credentials, API keys, and other sensitive information.

**Purpose:** Ensures that sensitive data such as credentials, tokens, and keys are

securely stored and managed, preventing unauthorized access.

**Features:**

**- Secret Storage:** Centralized storage for all types of secrets.

**- Access Control:** Granular access controls to ensure only authorized users and

systems can access secrets.

**- Audit Logs:** Detailed logging of access and modifications for compliance and

security monitoring.

**- Automated Secret Rotation:** Regularly rotates secrets to minimize the risk of

exposure.

**3.6. API Management**

* **Azure API Management (APIM):**

**- Description:** Azure API Management is a service that enables organizations to

publish, secure, transform, maintain, and monitor APIs.

**- Purpose:** Provides a central interface to manage APIs, including routing,

security, rate limiting, and analytics.

**- Features:**

**- API Gateway:** Acts as a gateway to route requests to backend services.

**- Security:** Offers security features like OAuth 2.0, JWT validation, and IP

filtering.

**- Analytics:** Provides insights into API usage and performance.

**- Developer Portal:** Enables API consumers to discover, use, and subscribe to

APIs.

**3.7. Environment Management**

* **Branch Management:**

**- DEVELOP BRANCH:**

**- Purpose:** Used for development and quality assurance (QA) environments.

**- Workflow:**

**-** Code is merged into the DEVELOP branch after passing CI checks.

**-** Deployments to DEV and QA environments are triggered

from this branch.

**- RELEASE BRANCH:**

**- Purpose:** Used for pre-production (PRE-PROD) and production (PROD)

environment deployments.

**- Workflow:**

**-** Once code is tested and ready for pre-production and production it is

merged into the Release branch.

**-** Deployments to the PRE-PROD and PROD environment are triggered

from this branch.

* **Environment Configuration:**

**- Development (DEV) Environment:**

**-** Used for initial development and testing by developers.

**-** Frequent deployments and testing cycles.

**- Quality Assurance (QA) Environment:**

**-** Used for testing by QA teams.

**-** Focuses on identifying bugs and ensuring the application meets quality

standards.

**- Pre-Production (PREPROD) Environment:**

**-** Used for final testing before production.

**-** Simulates the production environment to identify potential issues.

**- Production (PROD) Environment:**

**-** Live environment where the application is available to end-users.

**-** Deployments are carefully managed to ensure stability and reliability.

**3.8. COMPARISION BETWEEN DEPLOYING DIRECTLY THROUGH CODE AND DEPLOYING USING DOCKER FILE ON AZURE APP SERVICE**

|  |  |  |
| --- | --- | --- |
| **Aspect** | **Deploying Directly Through Code** | **Deploying Using Docker File** |
| **Configuration** | Limited to what Azure App Service  supports natively | Highly customizable with  Dockerfile instructions |
| **Portability** | Less portable; specific to Azure App  Service | Highly portable; can be deployed  anywhere Docker is supported |
| **Isolation** | Limited isolation; shares environment  with other apps | Full isolation; each app runs in its  own container |
| **Dependency**  **Management** | Less control over runtime environment can lead to inconsistencies if dependencies or environment settings change. | Complete control over  dependencies and their versions. Easier to manage dependencies and runtime configurations. |
| **Resource**  **Utilization** | Can be less efficient; shares resources  with other apps | More efficient; dedicated resources  per container |
| **Security** | Managed by Azure; limited control over  security policies | More control over security policies  and updates |
| **Flexibility** | Limited by Azure App Service's  supported languages and frameworks | Highly flexible; can run any  language or framework that  Docker supports |
| **Updates &**  **Rollbacks** | Easier to update code; rollback might  be more complex | Easy to update container; simple  rollback to previous image |
| **Debugging** | Easier with Azure's integrated tools | Requires container-specific  debugging tools |
| **Cost** | Potentially lower cost for small, simple  apps | Can be higher due to additional  overhead of managing containers |
| **Performance** | Can be limited by the underlying App  Service environment | Potentially better performance due  to optimized containers |

**3.9. REASON WHY WE HAVE TO DEPLOY OUR APPLICATION USING DOCKER IMAGE INSTEAD OF DEPLOYING DIRECTLY FROM CODE.**

**Advantages of Dockerfile Deployment**:

**Isolation and Consistency:**

**Environment Control:**

Each Dockerfile defines its own environment, ensuring consistency across development, testing, and production stages. This isolation helps prevent conflicts between different APIs and applications.

**Dependency Management:**

We can specify exact versions of dependencies in each Dockerfile, reducing the risk of dependency conflicts.

**Scalability and Flexibility:**

**Independent Scaling:**

Each API or application can be scaled independently based on its specific load and requirements, optimizing resource usage.

**Micro services Architecture:**

Dockerfile deployment aligns well with micro services architectures, where each service is independently deploy-able and manageable.

**Portability:**

**Cross-Platform Consistency:**

Docker ensures that the application behaves the same way regardless of where it is deployed (local machine, Azure App Service, Kubernetes, etc.).

**Simplified Testing:**

We can easily replicate production environments locally for testing purposes, ensuring more reliable deployments.

**Version Control and Rollbacks:**

**Image Versioning:**

Docker images can be versioned and stored in a container registry. This makes rollbacks straightforward, as you can deploy a specific version of an image if needed.

**Immutable Deployments:**

Docker images are immutable, meaning the deployment is predictable and reproducible.

**Disadvantages of Direct Code Deployment:**

**Environment Inconsistencies**:

**Shared Environment:**

Deploying directly from code to Azure App Service means all APIs and applications share the same environment, which can lead to conflicts and inconsistencies.

**Dependency Conflicts:**

Different APIs and applications might have conflicting dependencies, which can be challenging to manage in a shared environment.

**Limited Isolation:**

**Resource Contention:**

All applications run in the same environment, leading to potential resource contention and performance issues.

**Complex Configuration:**

Managing configuration settings and environment variables for multiple applications in a shared environment can become complex and error-prone.

**Conclusion:**

Dockerfile Deployment is the recommended strategy for our setup. This approach provides better isolation, consistency, and control over the environment for each API and web application. It also aligns well with modern development practices such as microservices, allowing for independent scaling and management of each component.

**Access List**

**For Ross Application**

1. BitBucket
2. Sync
3. Sonarcloud
4. Azure Container Registry
5. Azure App Service
6. Azure API Management
7. AKeyless
8. Azure DevOps
9. Azure Cloud Portal ( Subscription )