ARCHITECTURE

ARNAB CHAUDHURI

06-OCTOBER-2020

INSIDE

DESIGN FOR SCALE

USING MICROSOFT AZURE SERVICES

This whitepaper briefly walks through the various options available to design and architect a consumer-oriented Services and API platform, designed for scale using **MICROSOFT AZURE** Infrastructure as a Service and Native Platform as a Service offerings.

The whitepaper is not meant to be exhaustive, neither all the AZURE Services mentioned are mandatory to be used. AZURE Services mentioned are just suggestive and to develop a concrete architecture it requires deep investigation into existing line of business, technology landscape in use, scalability and high availability demands, organization budget and roadmap.

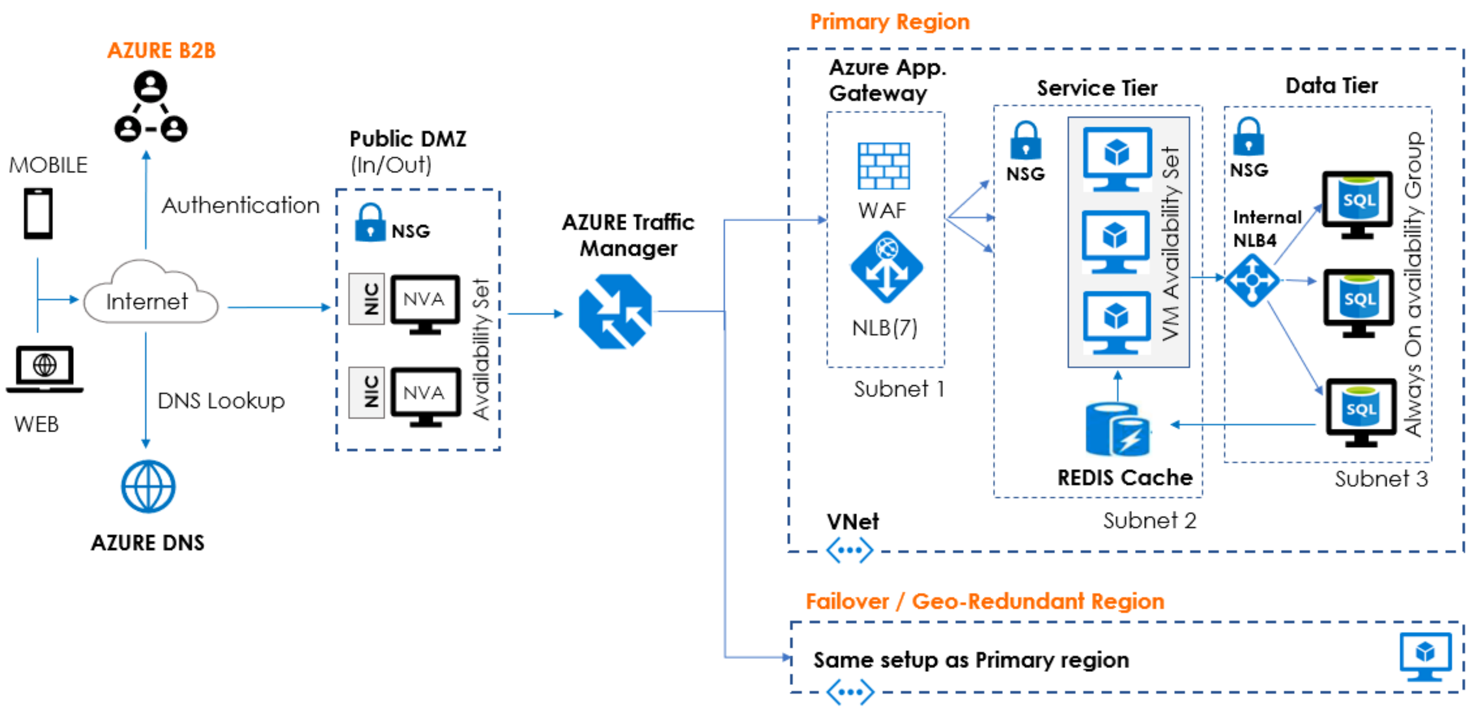
INFRASTRUCTURE AS A SERVICE (IAAS) SOLUTION

AZURE SERVICE: Azure App Service Environment

Architecture Principles: Traditional N-Tier Architecture

- A traditional three-tier application with a presentation tier, a middle tier, and a database tier.
- The N-tier application can have a **closed layer architecture** or an **open layer architecture** or a mix of both.
- The closed layer tiers will only call the next layer immediately down.
- The open layer architecture tier can call any of the layers below it or make any remote/external service or API calls.

ARCHITECTURE



Architecture Components

- Place multiple VMs in an **availability set** or VM **scale set** to achieve failover/resiliency in case one/more VMs fail
- Use **Load balancers** to distribute requests across the VMs in a tier. A tier can be scaled horizontally by adding more VM instances to the pool. Avoid vertical scaling to avoid application restarts or VM warm-ups.
- Place each tier inside its own **subnet**, so that their internal IP addresses fall within the same address range.
- Apply **network security group (NSG)** rules and route tables to individual tiers.
- Use **forced tunneling** if inbound/outbound request needs to go through a DMZ Layer/ firewall.
- For Database use **SQL Server, using Always on Availability Groups** for high availability in a separate tier.
- For higher security, place a network **DMZ** in front of the application to implement security functionalities like firewalls and packet inspection. DMZ acts as a proxy to shield the internal network from external threats like DDoS and script attacks.
- Restrict direct **RDP** access to VMs that are running application code. Use a **jumpbox** to login to VMs
- In a Hybrid scenario, extend the Azure virtual network to your on-premises network using a **site-to-site virtual** private network (VPN) or Azure **ExpressRoute**.
- Avail higher availability and disaster recovery by replicating the application across two/more regions and use **Traffic Manager** for failover based on "priority" settings.

Best Practices

- Use autoscaling (Metrics or Schedule based) to handle unpredictable changes in request load.
- Use asynchronous messaging to decouple tiers.
- Cache static and semi-static data which allows eventual consistency than strong consistency.
- Configure database tier for high availability, using a solution such as SQL Server Always on Availability Groups
- Place a web application firewall (WAF) between the front end and the Internet traffic
- Place each tier in its own subnet and use subnets as a security boundary.
- Restrict access to the data tier, by allowing requests only from the middle tier(s).

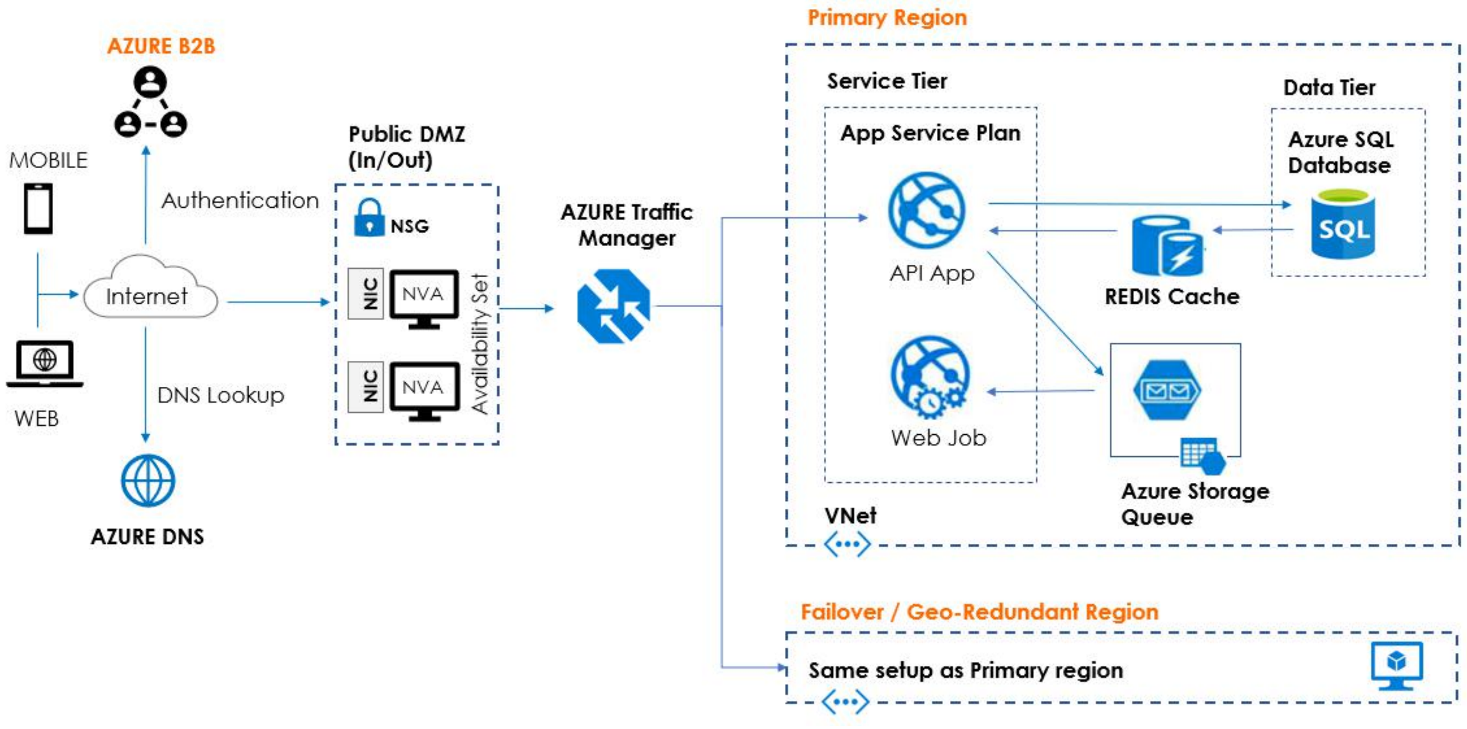
PLATFORM AS A SERVICE (PAAS) SOLUTION

AZURE SERVICE: Azure Web Apps with Azure WebJobs

Architecture Principle: Web-Queue-Worker

- A **web front end** that serves client requests
- A **worker process** that performs resource-intensive tasks, long-running workflows, or batch jobs.
- A **Message Queue** to bridge communication between the web front end and the worker
- One or more **Databases**.
- A **globally distributed cache** to store values from the database and user sessions.
- A **Content Delivery Network (CDN)** to serve static content (images/media etc.) based on user's geography

ARCHITECTURE



Architecture Components

- **App Service Plan**: Provides VM instances to Web App
- Application Service: Azure App Service **Web App**
- Worker Service: Azure **WebJob**.
- Message Queue: **Azure Service Bus** or Azure Storage queues (depending on size & volume of message)
- Use **Application Gateway** for SSL offloading and encrypted end-to-end secured traffic. Enable **Web Application Firewall** inside Application Gateway to protect from cross site scripting, session hijacking and DDOS attacks.
- Use route-based rules in **Application Gateway** to direct traffic to Image or Files VM instances.
- Use Azure **Redis Cache** to store session state and low latency data access.
- Use Azure **CDN** to cache static content (images, CSS, or HTML).
- Database: **Azure SQL Server**

Best Practices

- Expose a well-designed API to the client.
- Not every transaction has to go through the queue and worker to storage. The web front end should perform simple read/write operations directly. Workers should be used for resource-intensive tasks only.
- Set Autoscaling feature to handle changes in unpredictable user request load.
- Cache semi-static & near-static data.
- Use a CDN to host static content.
- Use polyglot persistence when appropriate.
- Partition data and setup local/geo-redundancy to improve scalability, reduce contention, and optimize performance.

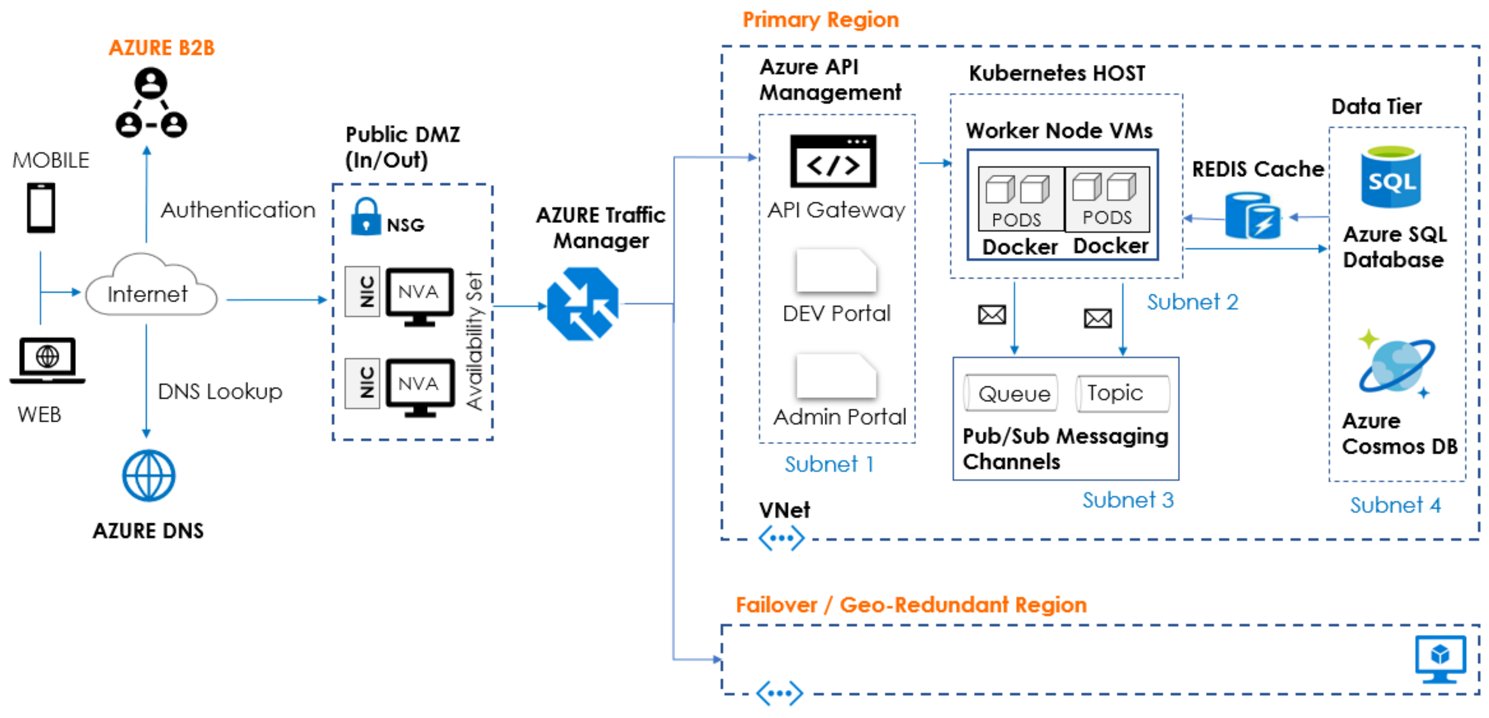
CONTAINER AS A SERVICE (PCAAS) SOLUTION

AZURE SERVICE: Azure Container Services

Architecture Principle: Microservices / Domain Driven Design Pattern

- Services should be small, a defined domain/subdomain in a **bounded context**, independent and self-sufficient.
- Each service should use a **separate codebase but in a single repository**, managed by multiple teams and may not necessarily share the same technology stack, libraries, or frameworks.
- Services can be **deployed independently**. Dependencies across services should be **isolated**. A team can update an existing service without rebuilding and redeploying the entire application.
- Services are responsible for **persisting their own data** (private to the service) or external state.
- Services **communicate** with each other and data stores using secured inter-service communication channels (listed later).
- **Isolate failures.** Use resiliency strategies, like circuit breaker patterns to prevent failures within a service from cascading.

ARCHITECTURE



Architecture Components

- Plan stateful or stateless Microservices based on usage and transaction demands.
- Use **API Gateway** for service entry point from external interfaces or client systems. API Gateway acts as a façade to help in routing, protocol conversion, legacy adapters, payload transformation and service catalog.

  Azure API Management can be used to manage all APIs in single place, secure & optimize them. Special attention is needed for scaling of API Gateway to avoid being it becoming a single point of failure.
- Services should be hosted in **Docker** containers managed inside a **Kubernetes Cluster**.
- Mount container **volumes** in Host system, for short term data persistence. Long term storage should be done in Databases.
- Use **Azure Container Registry** for managing Container Images & **Azure Kubernetes Services** for container management, service discovery, orchestration, and failovers.

- Azure Kubernetes Cluster (worker nodes) should be hosted in **Azure VNet** with proper Network Security Group (NSG) rules.
- **Interservice communication** should follow the **Event Driven Design** pattern using a Message Broker (**Azure Service Bus** – **Queue/Topic or Rabbit MQ**) to facilitate asynchronous communication. The following principles are suggested -
  - All Services send notifications to central Event Bus
  - There is one notification queue per service. Each service queue is subscribed to the Bus
  - Each service runs a background worker that listens to respective event queue
  - If an error is raised on a service, the message remains in queue for a default period (depending on the message queue used), and to be picked up later automatically when the error is resolved.
- Service **Data Aggregation** can follow either of the below patterns, depending on complexity of data interdependencies, read/write model, concurrency & data consistency guarantee needed.
  - **API Composition** – The API Gateway plays dual role of accepting incoming traffic as well as a data aggregator
  - **Choreography Pattern** – Another Service within the same cluster does the aggregation and can scale independently
  - **CQRS** – Split the read/write model if read volume is higher than writes. Allow synching between the separate data models. The read model can be materialized views to be kept up to date by subscribing to the stream of events emitted when data changes from the write model.
- Containers are processes and not durable in nature, hence **sessions** are to be managed out of the Host in **Redis Distributed Cache**, so that it persists even if the containers are rolled out or replaced. **Redis** allows global scalability.

Best Practices

- Stateless microservices are most recommended as they are lightweight in nature. However, they impose latency between the process & data sources as more state information needs to be passed between transactions. Performance improvement using caches and queues can lead up to complex architecture and needs to be well planned.
- Decouple producers(source) and consumers(target) services.
- No point-to point-integrations (except for smaller notifications). Flexibly add new services to the system.
- Consumer services can respond to events immediately as they arrive or put a blocking wait on the reply queue.
- Scale source or target services based on application demand.
- Transmit & monitor telemetry (perfmon counters – CPU, Memory etc.) data using Application Insights container in Host.

APPLICATION DESIGN PRINCIPLES

DEFINE "ABILITY(S)"

- **Scalability**
  - **Capacity:** Analyze and scale individual layers as demanded. Scale-down when not needed.
  - **Platform / Data:** Design based on small instances. partition data to aid scalability within persistence platform constraints (maximum database sizes, concurrent request limits, etc.). Collapse tiers to minimize internal network traffic
  - **Load:** Avoid contention issues and bottlenecks. Use asynchronous data exchange to help balance load at peak times. Use Azure load balancing features like Traffic manager, Azure Load Balancer etc. wherever needed.
- **Availability**
  - **Uptime Guarantees**: Ensure composite SLA Levels offered by provider meets the business guidelines.
  - **Replication and failover**: Map key application tiers/areas for redundancy and failovers. Ensure to replicate across separate geographies. Implement resiliency patterns like retry policy of circuit breaker for services
  - **Disaster recovery**: Define Recovery Time Objective (RTO) and Recovery Point Objective (RPO) with acceptable loss of data / business during a disaster. Plan backup strategy using Azure Backup and Site Recovery.
  - **Performance**: Implement performance quantifying measures and define acceptable level of performance drops. Implement monitoring for parts of the system that are the mostly highly contended. Implement auto-scaling and asynchronous communication.
  - **Security**: Adhere to data jurisdiction laws. Restrict access to data. Handle access policies and user lifecycle.
- **Manageability**
  - **Monitoring:** Use Azure monitoring tools. Generate metrics for logs and telemetry data. Implement auditing, compliance rules, whitelisting of APIs and data governance policies.
  - **Deployment** – Automate releases and rollbacks. Setup multiple environments and test suites between environments. Manage environment variables in Azure configurations. Setup environment slots to induce blue/green deployments.

API / SERVICE DESIGN GUIDELINES

- Pass only necessary parameters, restrict passing parameters not used. Return only necessary data
- Define only the Errors necessary. Distribute validation responsibility between client and services.
- Eliminate side effects, API calls should not do more than expected operations.
- Restrict functions to cover up negative actions.
- If response of API is huge or returning a lot of data, do a pagination or fragmentation of the API

USE 12 Factor App
PRINCIPLES

SECURITY AND QUALITY CONTROL

Concern	Where / How to Manage
Authentication	Azure B2B for external users. Azure AD Passthrough authentication or ADFS for internal users.
Authorization	▪ Define roles and role-based access **(RBAC)** ▪ Map roles to API ▪ Program API to deal with roles
Auditing & Validation	Enable auditing for incoming requests ▪ Input Validation ▪ URL Validation ▪ Output Encoding ▪ HTTP Status Codes
Message Confidentiality	SSL based end to end encryption through Internal **Load Balancer** or **Application Gateway**
Message Integrity	Deploy encryption and decryption for confidential parameters
External Threats - SQL Injection, DDoS attacks, Cross Site Scripting etc.	**Azure Application Gateway** with Web **Application Firewall** enabled

DATA DESIGN PRINCIPLES

SELECTING THE RIGHT DATA STORE

SELECT RDBMS – Azure SQL in VM (IAAS) OR Azure SQL Managed Instances (PAAS)

If Workload Includes	OR Data Types Are
• Both the creation of new records and updates to existing data happen regularly. • Multiple operations have to be completed in a single transaction. • Requires aggregation functions to perform crosstabulation. • Strong integration with reporting tools is required. • Relationships are enforced using constraints. • Indexes are used to optimize query performance. • Allows access to specific subsets of data.	• Data is highly normalized. • Database schemas are required and enforced. • Many-to-many relationships between data entities. • Constraints are defined in the schema and imposed on any data in the database. • Data requires high integrity. Indexes and relationships need to be maintained accurately. • Data requires strong consistency. Transactions need 100% consistent for all users & processes. • Size of individual data entries is small or medium-sized.

SELECT NoSQL – Azure Cosmos DB

If Workload Includes	OR Data Types Are
• Both the creation of new records and updates to existing data happen regularly. • No object-relational impedance mismatch. • Documents can better match the object structures used in application code. • Optimistic concurrency is more commonly used. • Data requires index on multiple fields. • Individual documents are retrieved and written as a single block.	• Data can be managed in de-normalized way. • Size of individual document data is relatively small. • Each document type can use its own schema. • Documents can include optional fields. • Document data is semi-structured, meaning that datatypes of each field are not strictly defined. • Data aggregation is supported.

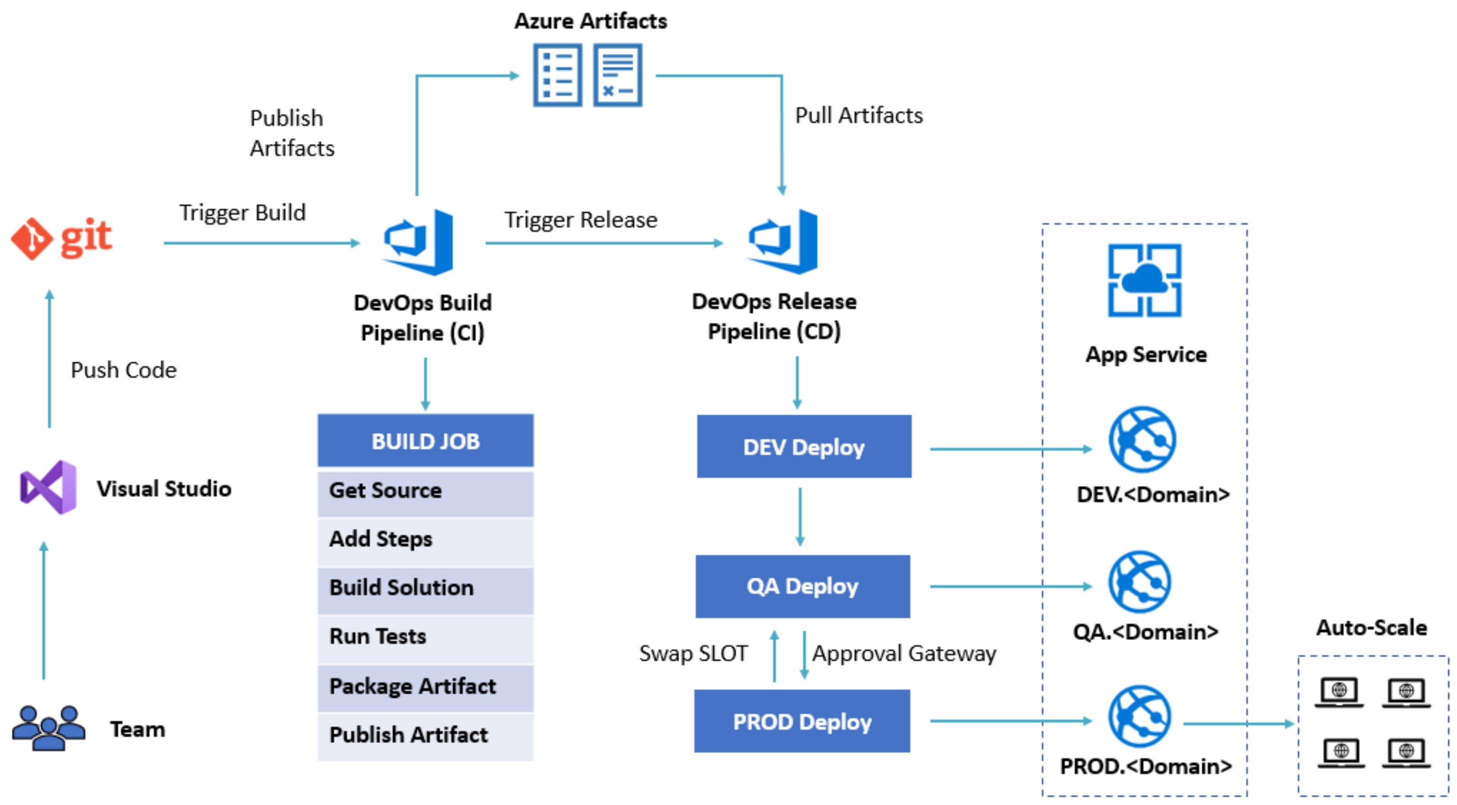
SECURITY AND QUALITY CONTROL

Concern	Where / How to Manage
Authentication	▪ Logins and authentication in SQL Server
Authorization	▪ Fixed server and database roles, custom database roles, and built-in accounts ▪ Object ownership and user-schema separation ▪ Granting permissions using the principle of least privilege
Governance	▪ Deploy runtime governance to ascertain request thresholds, and requests queues.
Data Security (SQL Server)	▪ Database **Roles** and Permissions ▪ Transparent Data Encryption **(TDE)** for Data at Rest ▪ Always Encrypted (Column-Level Encryption – **CLE**) for sensitive data in movement ▪ Dynamic Data **Masking** – To obfuscate sensitive data for queries ▪ Server-level **firewall** rules ▪ Database-level **firewall** rules

BUILD/DELIVERY PIPELINES TO SCALE RELEASES - 1

AZURE SERVICE: Azure DevOps Services (previously VSTS)

PLATFORM AS A SERVICE CI/CD PIPELINE



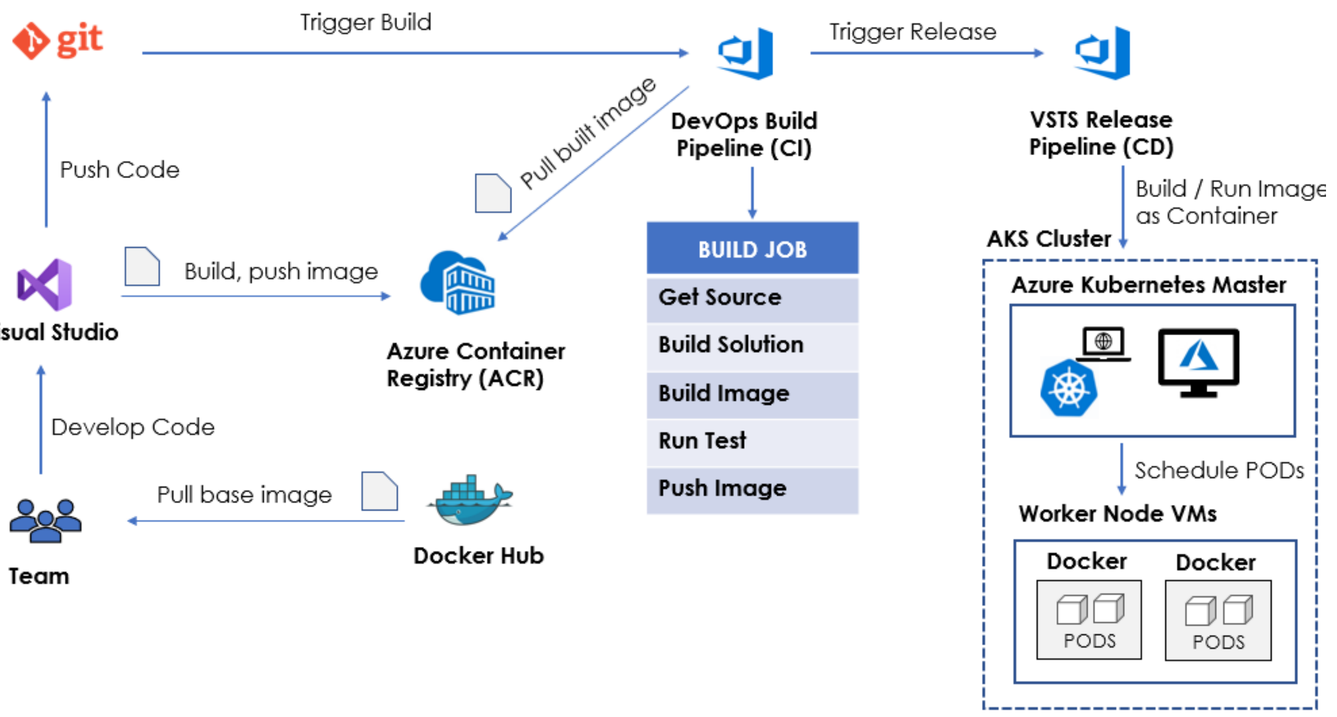
Best Practices

- Use a private organization repository
- Add multiple teams in a single project (avoid using multiple projects)
- To support continuous integration and deployment, add necessary agents, agent pools, and deployment pools
- Customize project dashboards for each team to share information and monitor progress
- For each Git repository, apply branch policies and define branch permissions.
- Customize your build and release pipelines, define build steps, release environments, and deployment schedule
- Define and configure test plans, test suites, and test cases as well as configure test environments; additionally, add test steps within your build pipelines
- Create separate build and release pipelines for DEV, TEST, PRODUCTION environments
- Implement Blue/Green deployments.
- Use SLOTS for deployments. Add steps for UAT / STAGING approvals before swapping SLOTS for Production

BUILD/DELIVERY PIPELINES TO SCALE RELEASES - 2

AZURE SERVICE: Azure DevOps Services (previously VSTS)

CONTAINER AS A SERVICE CI/CD PIPELINE



Best Practices

- **Scan for and remediate image vulnerabilities** - Only deploy images that have passed validation. Regularly update the base images and application runtime, then redeploy workloads in the AKS cluster
- **Automatically trigger and redeploy container images when a base image is updated** - Use automation to build new images when the base image is updated. As those base images typically include security fixes, update any downstream application container images.
- Include Test Suites in CI Build pipeline, monitor test results.
- **Continuous Integration (CI)** - Manage last known good version of successfully built images in Azure Container Registry
- **Continuous Delivery (CD)** – Always take the image successfully built and tested from CI Pipeline.
- Implement Blue/Green deployments
- Use SLOTS for deployments. Add steps for UAT approvals before swapping SLOTS for Production

EMPLOYEES / INTERNAL USER ACCESS (AZURE AD)

AZURE SERVICE: Azure AD Cloud / On-Premise AD Domain Accounts

OPTIONS

1. **Azure AD Cloud Account**
   Create traditional username and password identity management with roles and permissions management. Use App Registration secure your apps and associate/connect to Active Directory for authentication. Add SSO to your app, allowing it to work with a user's common credentials.
2. **Azure AD Synched Account (using Azure AD Connect)**
   - Azure AD Password Synchronization
     Hash synch On-Premise password using AD Connect sync. Setup Seamless SSO
   - Azure AD Passthrough Authentication
     Setup dual access to both Cloud & On-Premise resources using single credentials using Authentication Agent and a persistent connection. Setup Seamless SSO or MFA.
3. **Azure AD Federated Identity**
   Setup ADFS Federation Server, Proxy Server, ADFS Web Server. Setup trust with On-Premise Domain Controller. Allow users to connect to Azure or On-Premise systems using their Azure AD or Office 365 Accounts. Setup Seamless SSO
4. **Azure AD Domain Services**
   Create a managed domain within the Azure AD tenant. Add On-Premises or Azure VMs to join this managed domain and use the usernames, passwords, and group memberships from Azure AD to log in or authenticate.

SUGGESTED APPROACH

- **If Azure Tenancy is new and storing hashed credentials in Cloud allowed**
  - Use Azure AD Password Synchronization + Seamless SSO

    Decision Driver – If you want Azure to natively handle sign-in completely in Cloud and you do not want to enforce user level Active Directory Security Policies during sign-in.
- **If Azure Tenancy is new and storing hashed credentials in Cloud is restricted (compliance guidelines)**
  - Use Azure AD Passthrough Authentication + Seamless SSO

    Decision Driver – If you want Azure to handle sign-in with Cloud and you want to enforce user level Active Directory Security Policies during sign-in. If you do not want sign-in disaster recovery or leaked credentials report.
  - Use Azure AD Passthrough Authentication + Seamless SSO with Password Hash Synch.

    Decision Driver – If you want Azure to handle sign-in with Cloud and you want to enforce user level Active Directory Security Policies during sign-in. If you want sign-in disaster recovery or leaked credentials report.
- **If already an Office 365 tenant**
  - User Azure AD Federated Identity

    Decision Driver – If you do not want Azure to handle sign-in with Cloud and want to integrate with a existing federation identity provider. If you do not want sign-in disaster recovery or leaked credentials report.
  - User Azure AD Federated Identity with Password Hash Synch

    Decision Driver – If you do not want Azure to handle sign-in with Cloud and want to integrate with a existing federation identity provider. If you want sign-in disaster recovery or leaked credentials report.

PARTNERS / B2B ACCESS

AZURE SERVICE: Azure AD B2B

CHALLENGES

- Managing the lifecycle of Partner accounts. Controlling access to resources and applications.
- Monitoring account activities and granting access privileges.

OPTIONS

1. **Azure AD account for your tenant.**
   Create a **dedicated account** in your Azure AD tenant with a local username and password.
2. **Azure AD account in another tenant**
   Partners **might already have an Azure AD account**. Or create one in a **dedicated tenant** which might be extended to **invite XYZCORP** tenant.
3. **A consumer grade Microsoft Account**
   You can invite a user with a **Microsoft Account** to join your Azure AD tenant.

SUGGESTED APPROACH

1. Use organization **Azure AD accounts** to grant access to your Azure subscription to have more control over the accounts which are getting access to your Azure resources
2. **Optionally** enforce policies like **Multi-Factor Authentication**, conditional access & manage lifecycle of partner accounts using **Azure Privileged Identity Management**.
3. Allow partners to get full benefit of **SSO** using **Azure B2B**

- **If Partners do not have Azure or are not Azure Tenant from other organization**
  - **Create Dedicated tenant for partners** (if partners do not have AZURE) **- Establish a dedicated tenant** for managing external accounts – e.g. **partners.xyzcorp.com** – through the standard tenant handling the enterprise users for the domain **xyzcorp.com**.
  - Separating guest users from production tenant gives an **additional layer of control and a security boundary.** These users will not get same access level as the standard users in the organization. **XYZCORP** will have to invite them & assign access.
- **If Partners are Azure Tenants**
  - Similar as directly adding a user over a trust to the enterprise Azure tenant. It requires admin rights in both tenants. Log on to another tenant, pick up the user(s), getting back to your original tenant and creating a “reference” to the user.

INVITING PARTNERS TO USE B2B NETWORK (OPTIONS)

- Instead of creating a new set of individual accounts, invite users to a **target directory with XYZCORP accounts** using **Azure AD B2B** and grant access to the necessary resources.
- Bulk Invite user accounts by using **CSV File Upload** option.

CUSTOMERS / B2C ACCESS

AZURE SERVICE: Azure AD B2C

OPTIONS

1. **Work Account.**
   Users with work accounts can create new consumer accounts, reset passwords, block/unblock accounts, and set permissions or assign an account to a security group.
2. **Guest Account**
   Invite external users to your tenant as guests, share controlled administrative access.
3. **Consume Account**
   Invite users to complete the sign-up user journey in an application you have registered in your tenant.

IDENTITY EXPERIENCE

1. **User flows** – Use predefined, built-in, configurable policies that Azure provides to create sign-up, sign-in, and policy editing experiences in minutes.
2. **Custom policies** - Create your own user journeys for complex identity experience scenarios

EXTERNAL IDENTITY PROVIDERS

- Configure Azure AD B2C to allow users to sign-in to applications with credentials from external social or enterprise identity providers (IdP). Azure AD B2C supports external identity providers like **Facebook**, **Microsoft account, Google, Twitter**, and any identity provider that supports OAuth 2.0, OpenID Connect, and SAML protocols.

IDENTITY PROTOCOLS

- **Applications** - Azure AD B2C supports the OAuth 2.0, OpenID Connect, and SAML protocols for user journeys. Your application starts the user journey by issuing authentication requests to Azure AD B2C. The result of a request to Azure AD B2C is a security token, such as an ID token, access token, or SAML token. This security token defines the user's identity within the application.

- **External Identities** - Azure AD B2C supports federation with any OAuth 1.0, OAuth 2.0, OpenID Connect, and SAML identity providers.

SECURING IDENTITIES

- Multi-factor Authentication

- Automatic Account Lockouts

- Rule based password complexity

SINGLE SIGN-ON OPTIONS

- **Azure AD B2C** - Session managed by Azure AD B2C
- **Federated identity provider** - Session managed by the identity provider, for example Facebook, Salesforce, or Microsoft account
- **Application** - Session managed by the web, mobile, or single page application