Monthly Status Meeting March 2022

SAVVLY

ART SEDIGHI, PHD

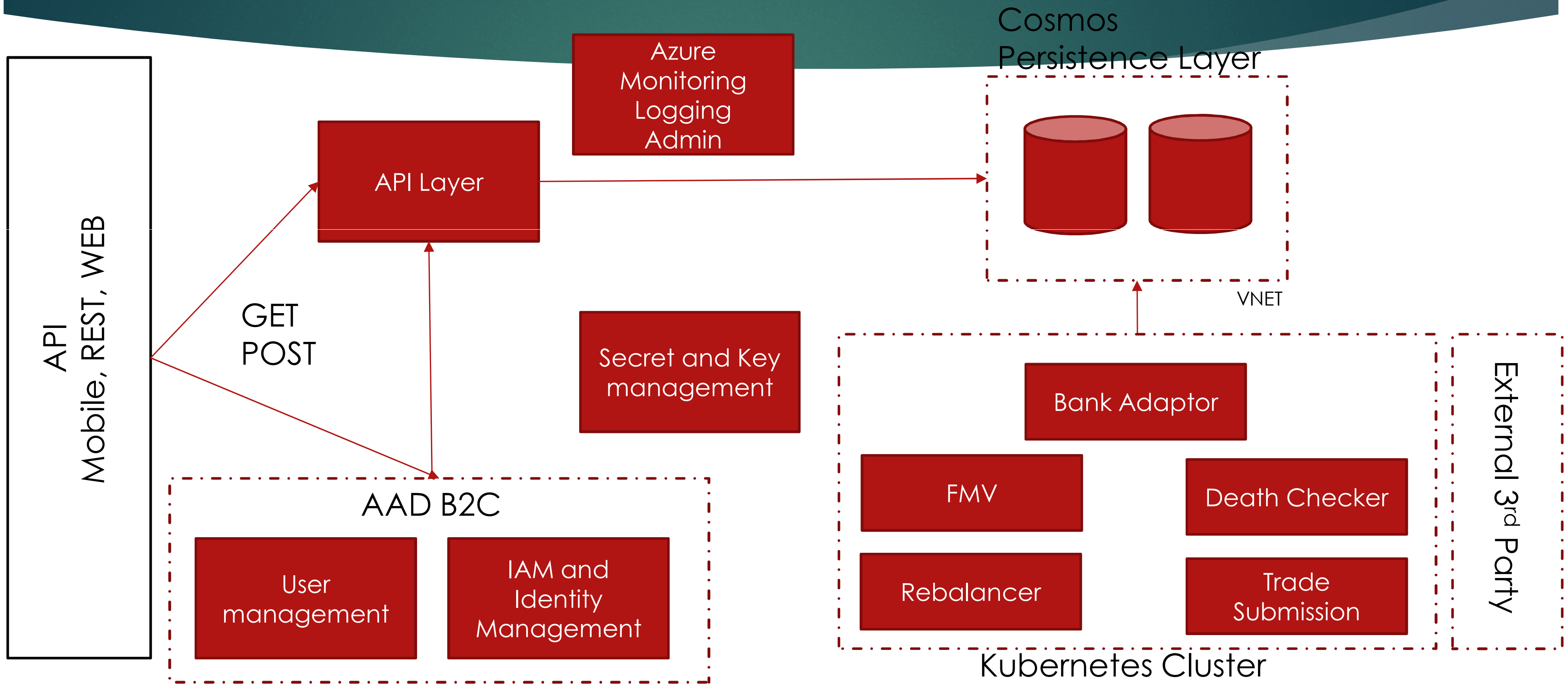
Overall Architecture

- Cloud-based infrastructure for all of SDLC and production
  - Segregated infrastructure for production with tight access control
- Multi-region deployment
  - The product calls for as many regions as possible
- Use of PaaS cloud services for the initial release
  - Engineering and development for later releases in order to increase performance and reduce cost

Overall Architecture

- n-tier architecture:
  - Clients: come from mobile devices or REST calls via Web
  - Server: business logic that manages requests
  - Persistence Layer: Cloud NoSQL storage vis CosmosDB and blob store as needed
  - Adaptors to external and third-party service providers
- Security architecture
  - VNETs for data, business logic and web
  - Application Gateway and Firewalls to control traffic from the public Internet
  - No internet access from business logic or backend store

Domain Architecture



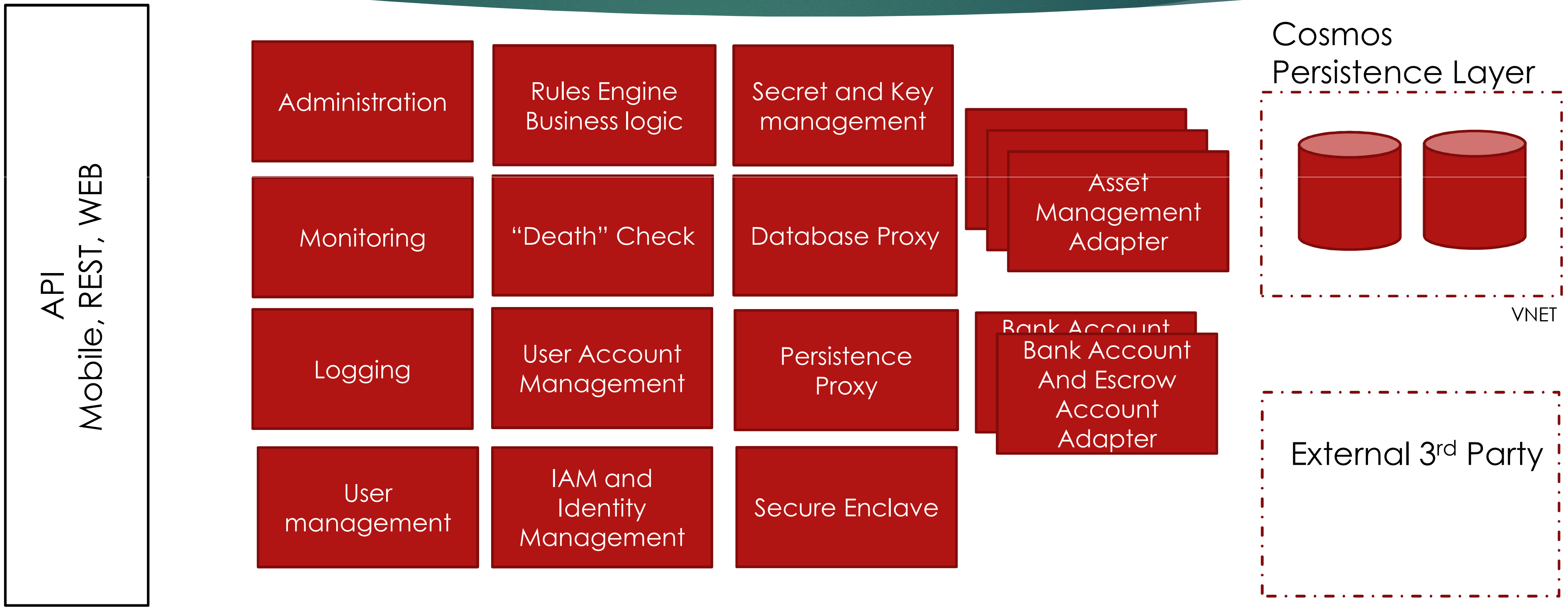
Timeline and Progress

	Feb				March				April				May				June				July			
API	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨
Secret Management	🟨	🟨	🟨	🟨	🟨	🟨				🟨	🟨	🟨												
Monitoring														🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨
User management	🟨	🟨	🟨				🟨	🟨	🟨															
Identity Management			🟨	🟨	🟨																			
Death Checker			🟨			🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨									
Bank Adaptor								🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨				
FMV						🟨																		
Rebalancer						🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨									
Trade Submission										🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨		
Web integration									🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨
Testing											🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨

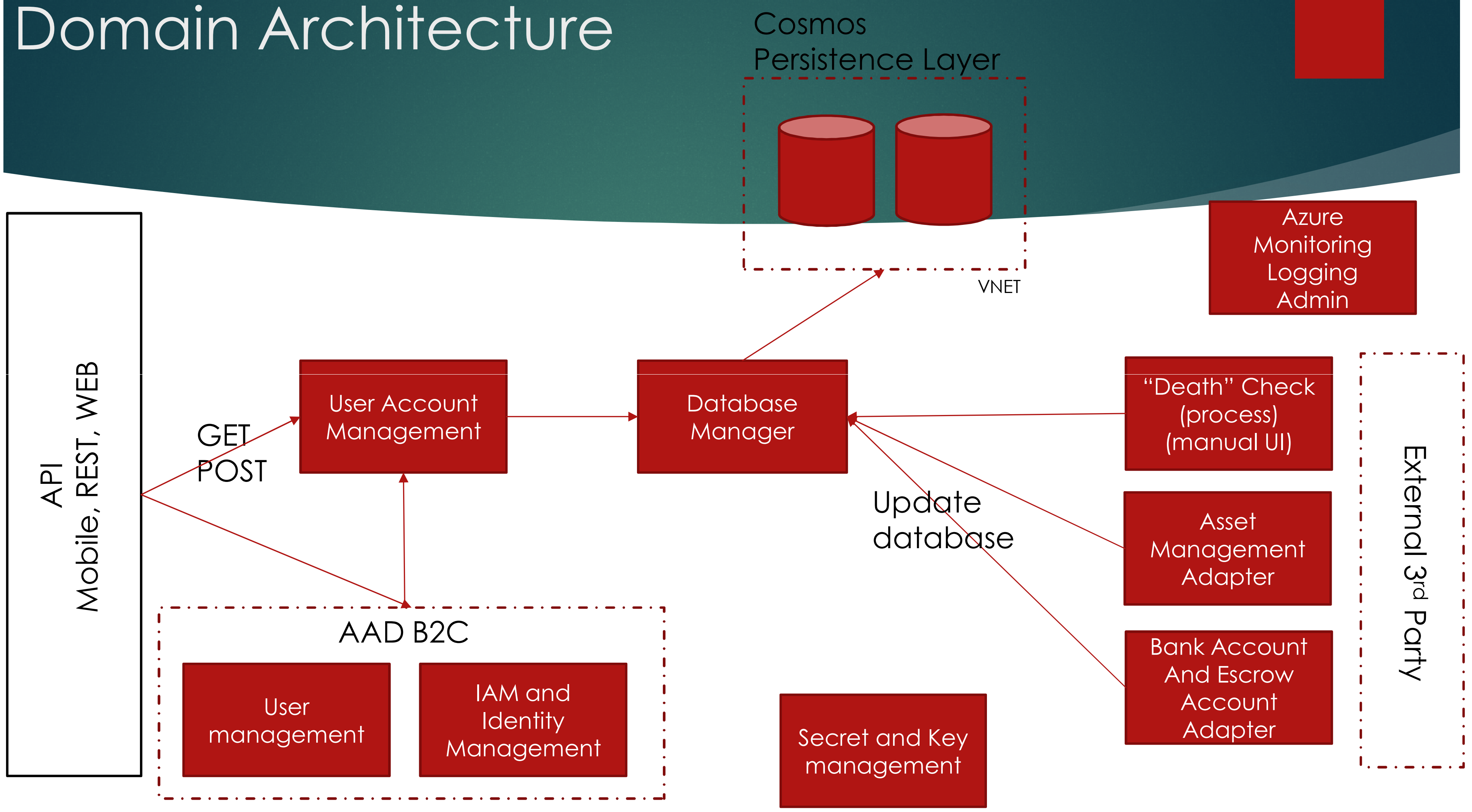
Resourcing

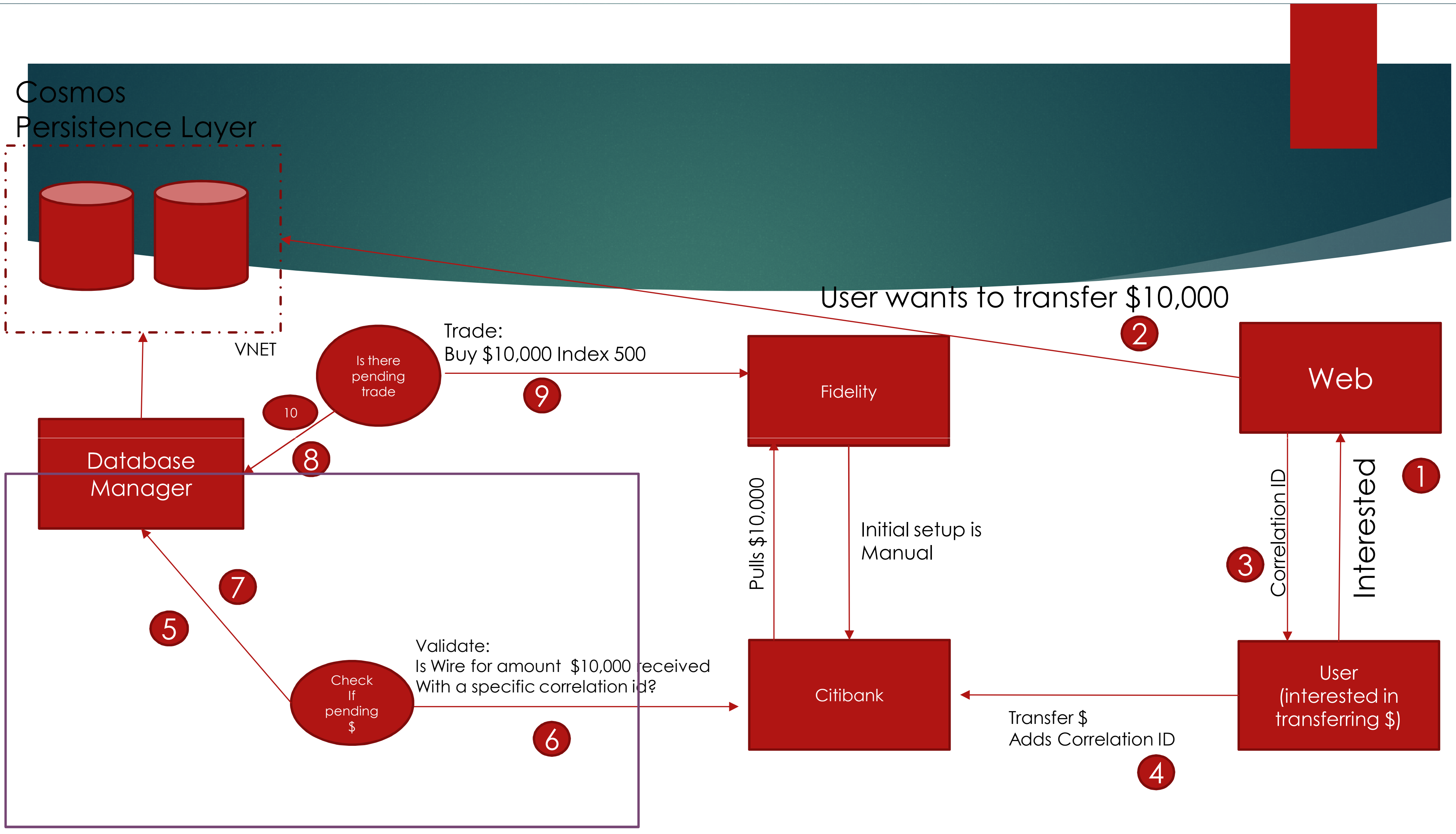
- Art - Cloud, FMV, Rebalancer
- Danny – API
- Dave – Cloud, Bank Adaptor
- Mainor – Death checker
- TBD – Identity, Secret, Trade submission
- TBD – Testing, Web integration, monitoring

Domain Architecture

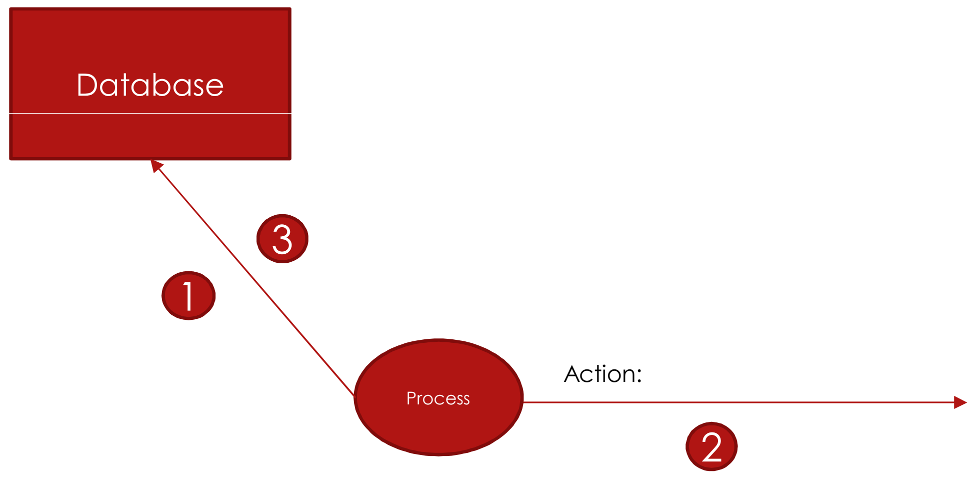


Domain Architecture




Cosmos
Persistence Layer
VNET
User wants to transfer $10,000
2
Web
Interested
1
Correlation ID
3
User
(interested in
transferring $)
Transfer $
Adds Correlation ID
4
Citibank
Fidelity
Pulls $10,000
Initial setup is
Manual
Trade:
Buy $10,000 Index 500
9
Is there
pending
trade
10
8
Database
Manager
7
5
Check
If
pending
$
Validate:
Is Wire for amount $10,000 received
With a specific correlation id?
6

Pattern



User account management + rules engine

- Rules engine is a standalone process
  - Checks the account of the user
  - if there needs to be rebalancing
  - if wire transfer is completed
  - Calculates the fmv of the portfolio every 24 hrs
  - Checks for wire transfer requests and checks for completion
  - Submits new trades

Rules engine could be divided into multiple micro-services:
- **Trades**: a service that just submits trades
- **Money transfer**: a service that just checks for pending and successful money transfer
- **Death**: a service that checks for "death" of a user
- **FMV**

User account management + rules engine

- User account management is flask based api
  - Interfaces with the database
  - Mostly read (GET)
  - A couple of POST calls:
    - Money wire request

Specific Components

- Azure Virtual Machines: hosts services and business logic
- Azure Active Directory: authenticator and user management
- Azure Security center: Security information and event management (SIEM)
- Azure Application gateway: traffic router, filter, firewall
- Azure Application server: main business logic, integration hub with other components
- Azure Key Vault: single entry secure vault to keep bank account information
- Azure CosmosDB: distributed database to hold shared transactions, contacts, and other system state
- Azure Storage: logging
- Azure Log Analytics: event management

Architectural Goals

- API driven architecture, use of micro-services and stateless design patterns
- Production environment is logically segregated from the rest of the resources on Azure
- All networking and communication is internal to Azure
  - Only one entry point to the environment (web 80/443) – everything else is blocked off
- All networking is segregated out to its own resource group and managed accordingly
  - Not affected by production resources
- Integration with external providers is decoupled to allow for ease of transition to other providers

Architectural Non-Functional Goals

- Primary: Security
- Secondary:
  - Cloud-native: API driven, micro-services
  - Abstraction of integration points
  - Performance

Engineering Plan

- Resource and Team buildout: Jan 31st (approx.)
- Start Date: Feb 1st (depending on resource allocation)
- Alpha release: May 31st (depending on start date)
  - End-to-end communication sand-box: start date plus 45-days
- Total resource requirement:
  - 4 Senior developers - full time
  - 1 Senior DevOps – half-time
- Estimated cost: 350k + Azure costs
- Not currently part of the engineering plan:
  - Web, Mobile and any client facing components

Appendix

Web Layer

- Application gateway is the main external entry to the environment
- Application gateway has a web application **firewall** (WAF) feature that is setup to prevent attacks that are outlined in the OWASP 10 documentation
- App Service Environment hosts the API, which is an isolated app server, and only hosts Savvly product (**single tenant**, separate network and storage)
- The Azure App Service Environment is an Azure App Service feature that provides a fully **isolated** and dedicated environment for securely running App Service apps at high scale
- The app service further isolates network traffic by allowing traffic to only come from the app gateway. The ensures that the app service is further **isolated** from unwanted access
- All the connection to the application gateway are over **SSL**

Cosmos DB

- Azure Cosmos DB is a fully **managed** NoSQL database
- **Single-digit** millisecond response times, and automatic and instant scalability, guarantee speed at any scale
- Real-time access with fast read and write latencies globally, and throughput and consistency all backed by SLA with **99.999%** availability, and enterprise-level security
- The Cassandra API for CosmosDB enables interaction with data stored in Azure Cosmos DB using the Cassandra Query Language (CQL) , Cassandra-based tools (like cqlsh) and Cassandra client drivers that you're already familiar with

networking

- Azure Virtual Network (VNet) is the fundamental building block for private networks in Azure
- VNet enables resources to securely communicate with each other without ever going over the public internet
- The production environment further segregates traffic to the following vnets:
  - Key vault vnet: holds band information. This vnet has a dedicated access control, and firewall
  - Database vnet: all traffic to the database is segregated and firewalled. No traffic to the database is allowed unless it flows thru proper set of resources
  - Gateway subnet in vnet: composed of a dedicated subnet for the internet gateway. This is the only external facing IP
    - Firewall: controls traffic and prevents against OWAPS 10 based attacks
  - Application Server subnet in vnet: dedicated application server (not multi-tenant). Only traffic from the app gateway is allowed

Dedicated Network

- Private endpoint is a network interface that uses a private IP address from ‘a’ virtual network that connects privately and securely to a service
- By using a private endpoint, the service can only be accessed from the virtual network
  - Private endpoint basically allows connection to the resource only via the private end point route, but by putting the private endpoint in a vnet, we limit access to the resource
- Private endpoints in the architecture:
  - KV – connection to the key vault must flow thru the vnet, and then the private endpoint
  - Database (Cosmos) – connection and interaction with the database must flow thru the vnet, and then the private endpoint