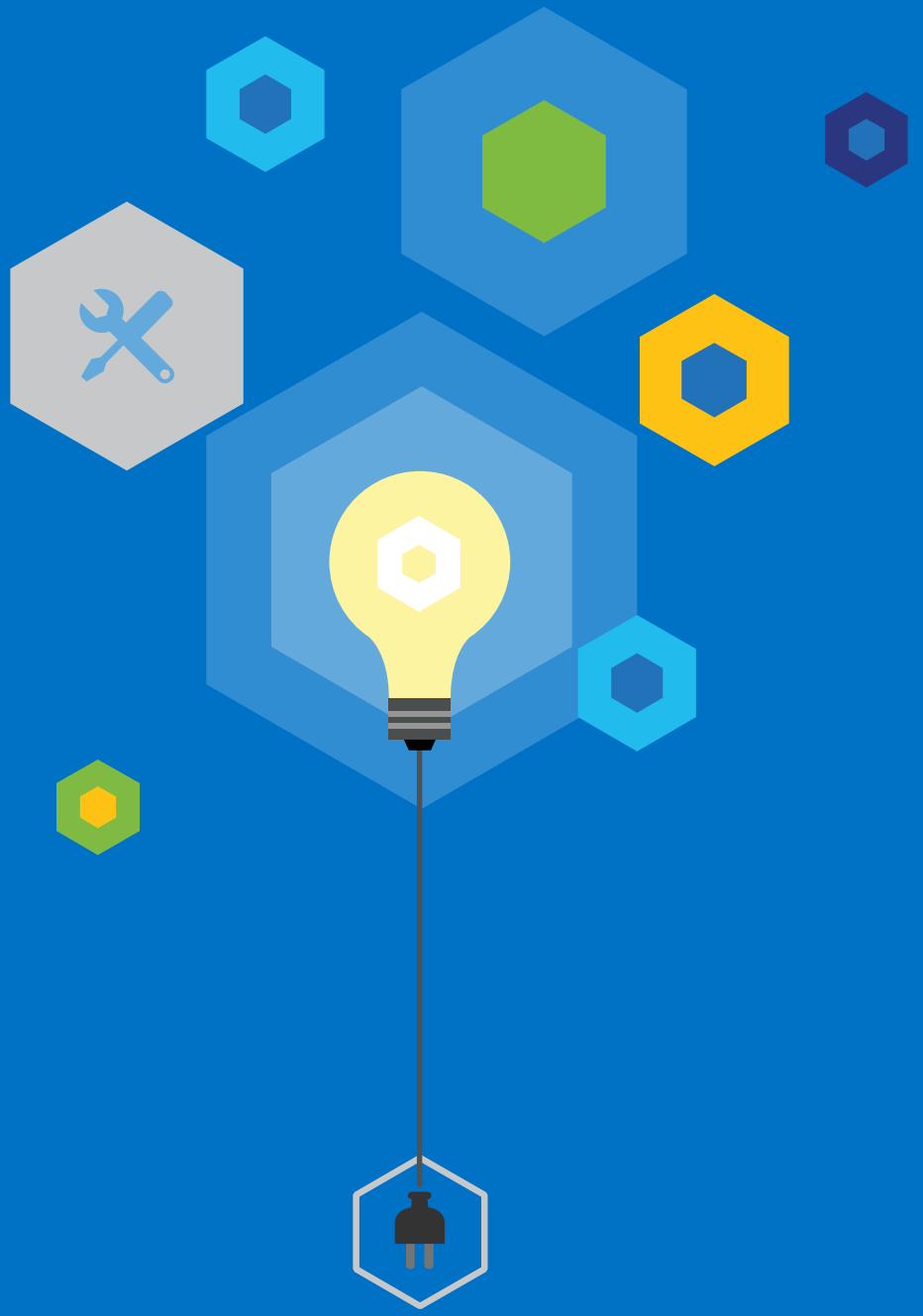
Microsoft Cloud's Frontdoor: Building a Global API

Charles Lamanna
March 8th, 2016

Microsoft



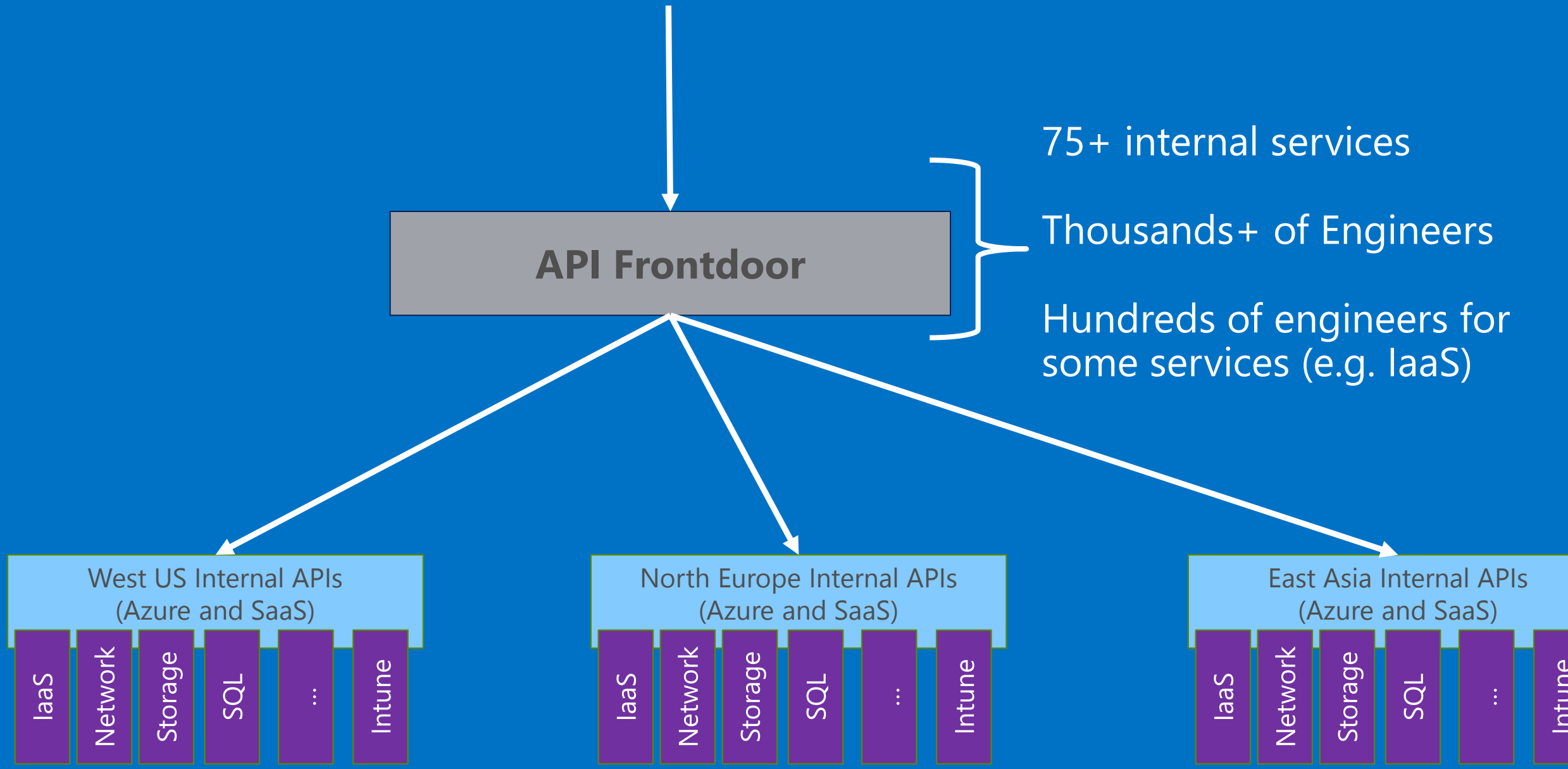
QUICK Intro

Group Engineering Manager at Microsoft

Joined when MetricsHub (co-founder) was acquired

API Frontdoor

- Acts as the gateway for Microsoft cloud services and products
- Performs common utilities (throttling, auditing, authN, authZ, etc.) and proxies to internal services
- Common pattern at large scale (e.g. Zuul / gateway from Netflix)


API Frontdoor
75+ internal services
Thousands+ of Engineers
Hundreds of engineers for some services (e.g. IaaS)
West US Internal APIs (Azure and SaaS)
IaaS
Network
Storage
SQL
...
Intune
North Europe Internal APIs (Azure and SaaS)
IaaS
Network
Storage
SQL
...
Intune
East Asia Internal APIs (Azure and SaaS)
IaaS
Network
Storage
SQL
...
Intune

Mission Critical

- "Hot path" for all cloud provisioning and management

- "Failure is not an option:" downtime must be avoided at all costs (otherwise business halts)

Downstream services are globally distributed

22 regions
5 continents

Our problem

.. but the gateway was hosted in a single region in the US (a SPOF)

Issues in the single region impacted all customer regions

3/3 Microsoft Azure classic portal - Multi-Region - Advisory

SUMMARY OF IMPACT: Starting at 03 Feb, 2016 11:47 to 15:29 UTC a subset of customers may have experienced latency and possible timeouts when attempting to login to their Classic Management Portal (http://Manage.WindowsAzure.com), and also when attempting to load/view their Azure resources from within their Management Portal. A subset of customers using PowerShell may have experienced limited latency. PRELIMINARY ROOT CAUSE: Storage latency increased due to an inefficient automatic scale-out operation, this in turn led to latency when performing read intensive operations primarily from the management portal. MITIGATION: Engineering manually applied a policy facilitate effective load balancing during the scale-out operation. Next Steps: Engineers will continue to investigate why the scale-out operation caused problems.

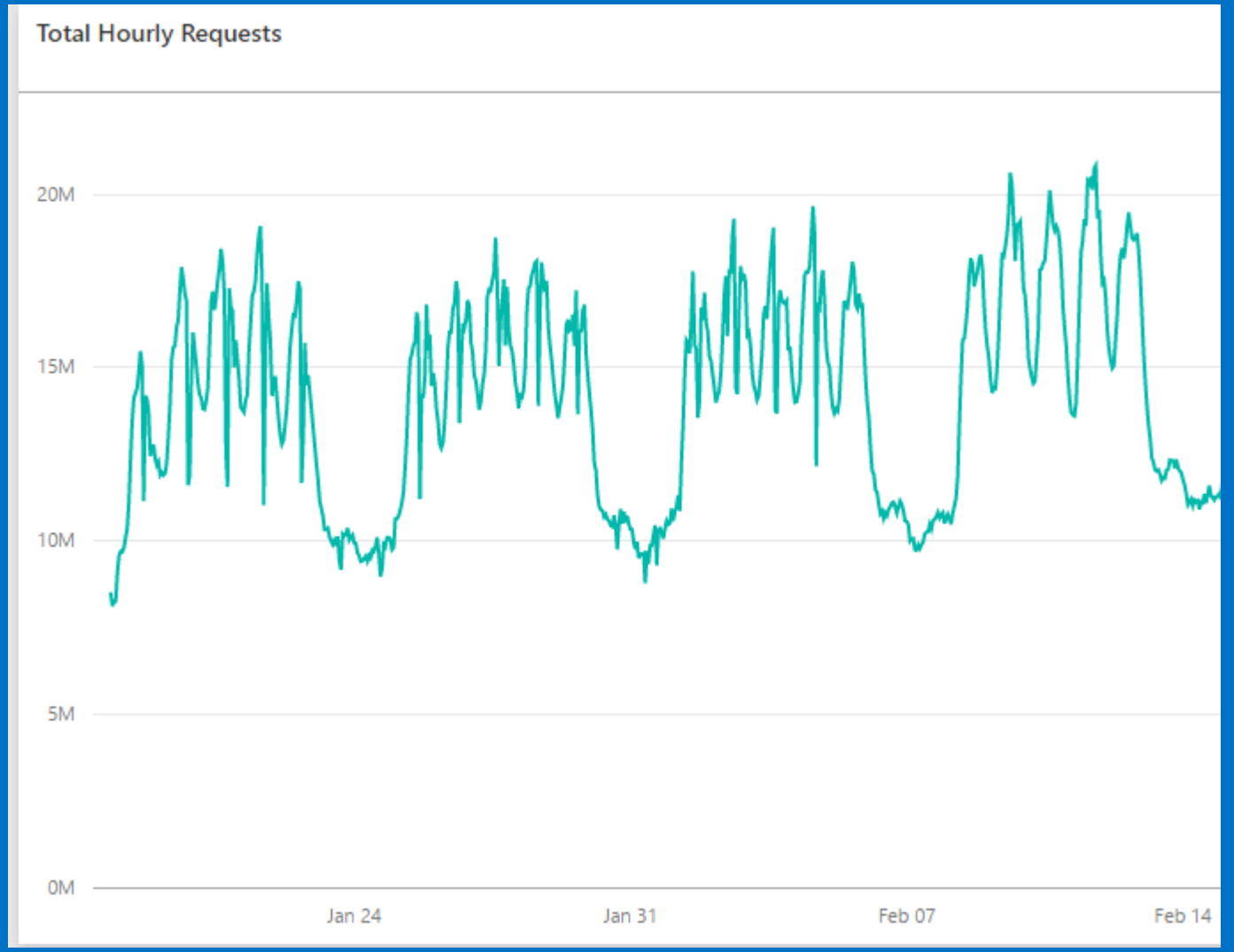
Data lives in a single region – caused data sovereignty issues

- Microsoft will not store customer data outside the customer-specified Geo except for the following regional services:
  - Cloud Services, which back up web- and worker-role software deployment packages to the United States regardless of the deployment region.
  - Azure RemoteApp, which may store end user names and device IP addresses globally, depending on where the end user accesses the service.
  - Preview, beta, or other prerelease services, which typically store customer data in the United States but may store it globally.

Usage was growing..

5,000+ API requests / s

20,000+ backend transactions / s



… and growing

- Huge growth in Azure usage (140% y/y revenue)
- 15-20% m/m growth in API requests
- API volume grows faster than rev/usage – more admins for same customer accounts, more monitoring / cost tools polling data, etc.

Friction for code velocity

- Increasing number of features / capabilities with a growing velocity

- Requires careful upgrades (node-by-node) to avoid impacting customers

We needed to..

1. build better resiliency into the service (app, data & processes)
2. make APIs work better around the world
3. maintain (and even improve) rate of new features

.. so, in 2013-2014, we re-architected the frontdoor

Principles

1. Keep it Simple

Keep it Simple

- Pride in a simple system – no value in complexity
- Outsource complexity to other services: Azure Storage, Software Load Balancer (SLB) and DNS
- Leverage OSS wherever possible

Be layer appropriate

- No paxos, no quorum, none of that

- HTTP + REST is good enough
- Standard MSFT development stack (C# + IIS)

- Keep compute stateless – push state into other services

2. Expect disaster

Bad things happen

- The scope of an issue and its frequency tend to be indirectly connected

- That does not change the fact that “huge” issues still happen

Everything will go wrong

- Every day: a node / VM has an issue
- Every month: a network / storage issue takes a region offline
- Every year: multi-region impact

3. People make mistakes

People are imperfect

- People can be distracted + need to sleep + make mistakes – but we need perfection 24x7

- Cannot have people involved in recovery or major, cross-region operations

No human involvement

- Ship several times a week – safe rollouts region-by-region

- Decision to continue with a deployment (or abort it), needs to be automatic

4. Don’t be afraid

Must avoid fear in the process

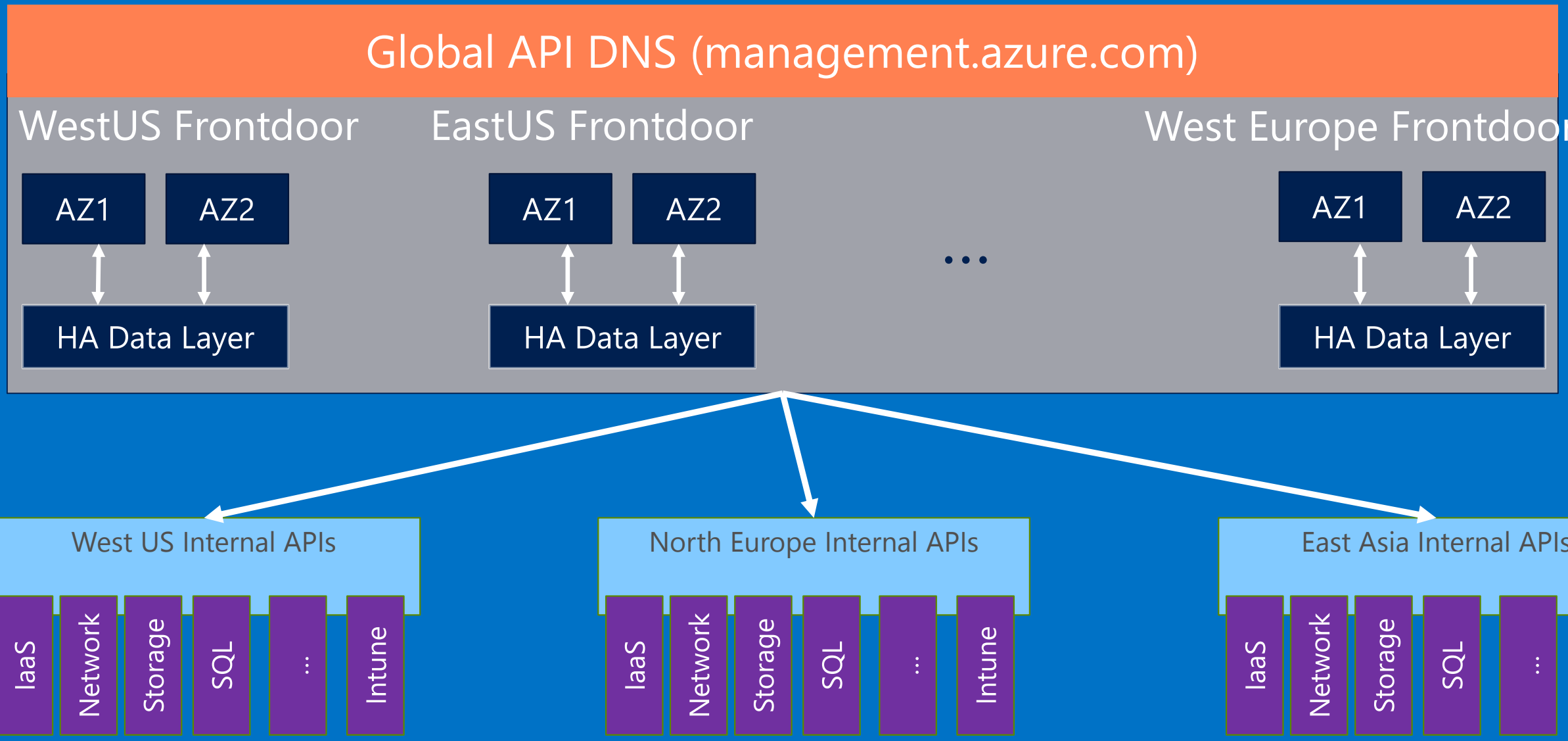
- Devs are also ops – not a separate team
- Need to build a system where we are not afraid of being on-call or making changes
- That means things like fail safes, circuit breakers, automatic failover, etc.

Consequences of fear

- Forces subjective speed vs. quality conversations
- If we do not have confidence in the system, the team will be paralyzed & the product is DOA

Architectural Details

What does a request look like?

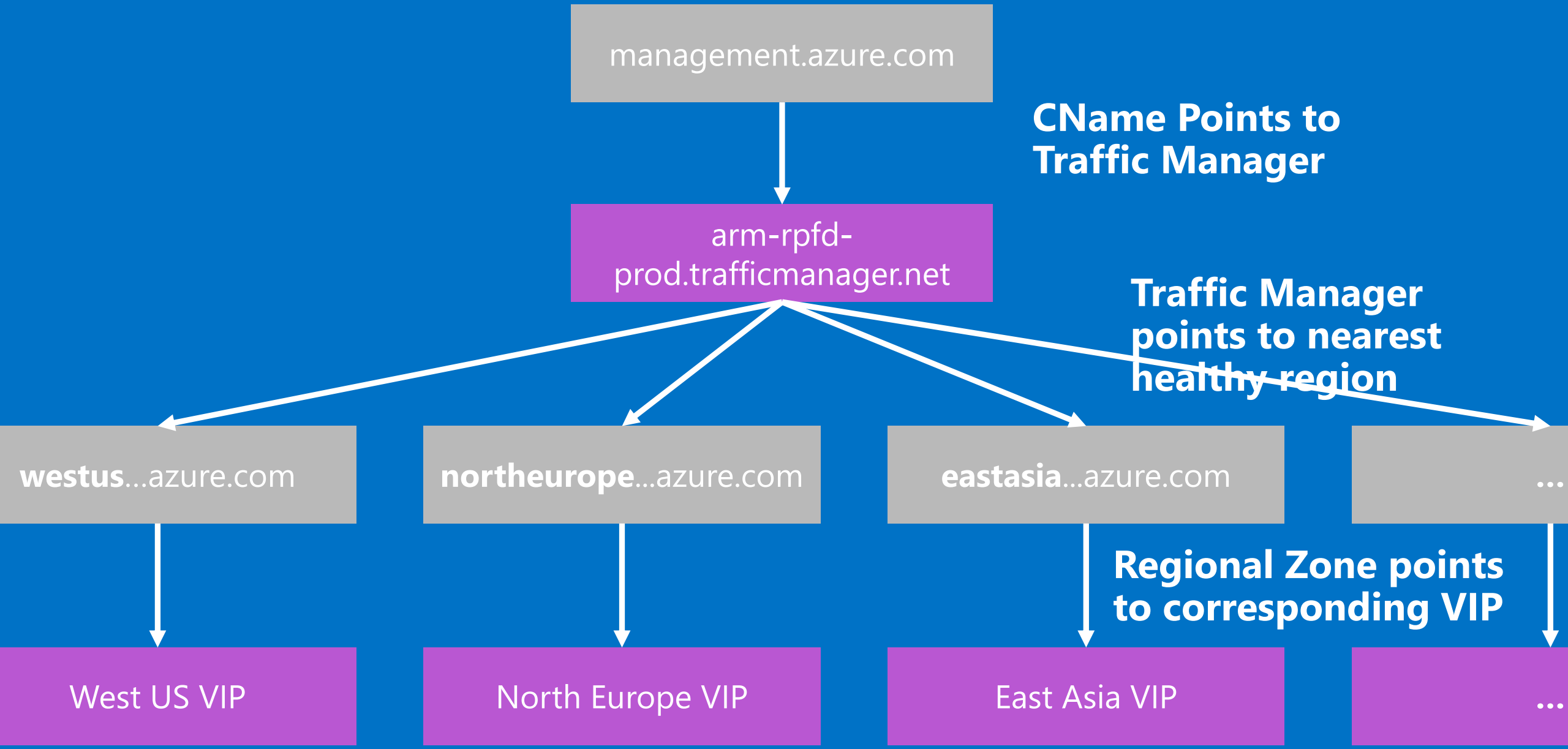


DNS Configuration

Layered DNS configuration

- Top level hostname for the API surface (management.azure.com)

- Hostname is resolved to nearest datacenter / region via Azure Traffic Manager (like Route 53 from AWS)
- Datacenter / regional hostname then points to the VIP


management.azure.com
CName Points to Traffic Manager
arm-rpfd-prod.trafficmanager.net
Traffic Manager points to nearest healthy region
westus...azure.com
northeurope...azure.com
eastasia...azure.com
...
Regional Zone points to corresponding VIP
West US VIP
North Europe VIP
East Asia VIP
...

Resolving hostname from hotel in London

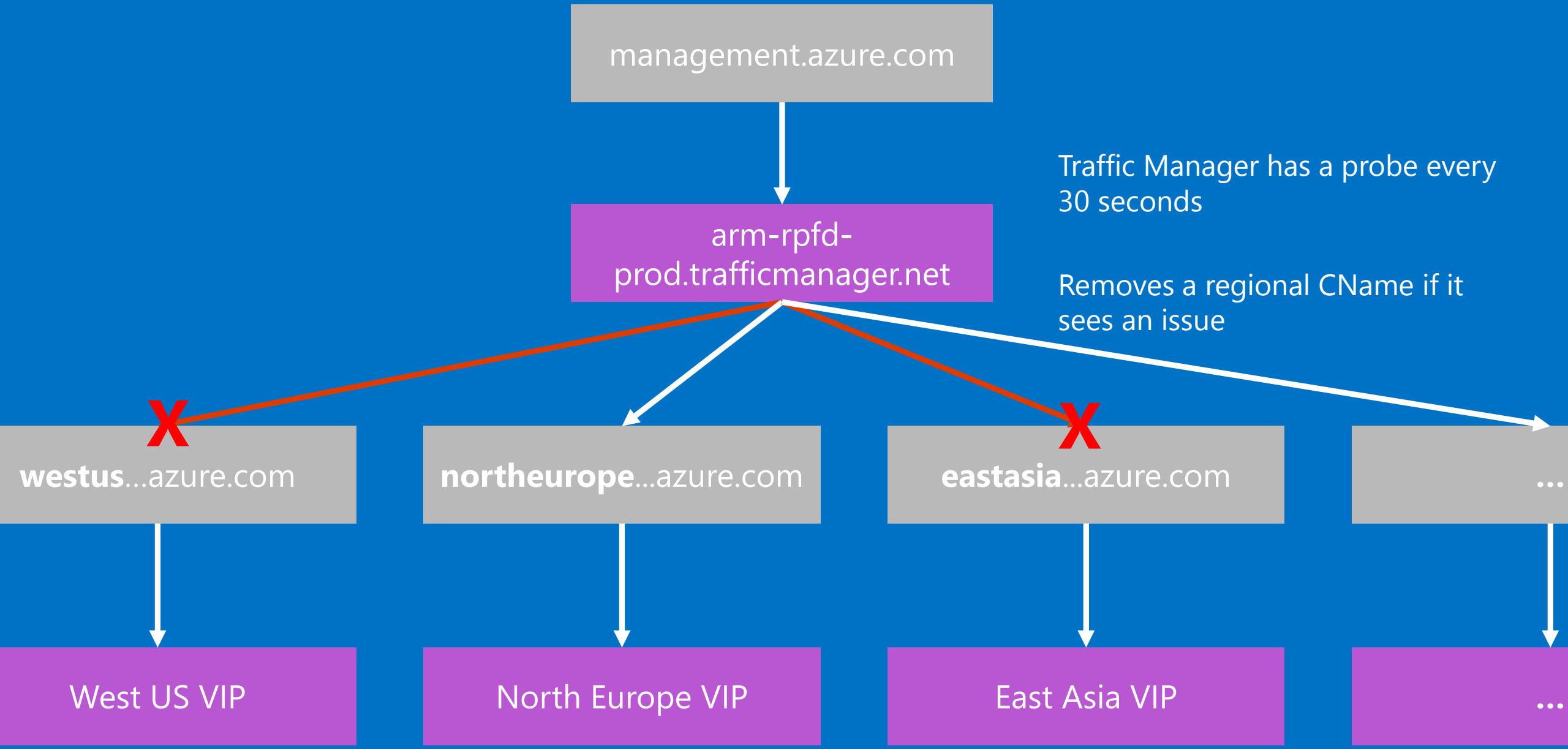
```
C:\>nslookup management.azure.com
Server:  UnKnown
Address:  10.211.55.1

Non-authoritative answer:
Name:    rpfd-prod-am-01.cloudapp.net
Address:  23.97.164.182
Aliases:  management.azure.com
          arm-rpfd-prod.trafficmanager.net
          westeurope.management.azure.com
```

Resolving hostname from VPN in US

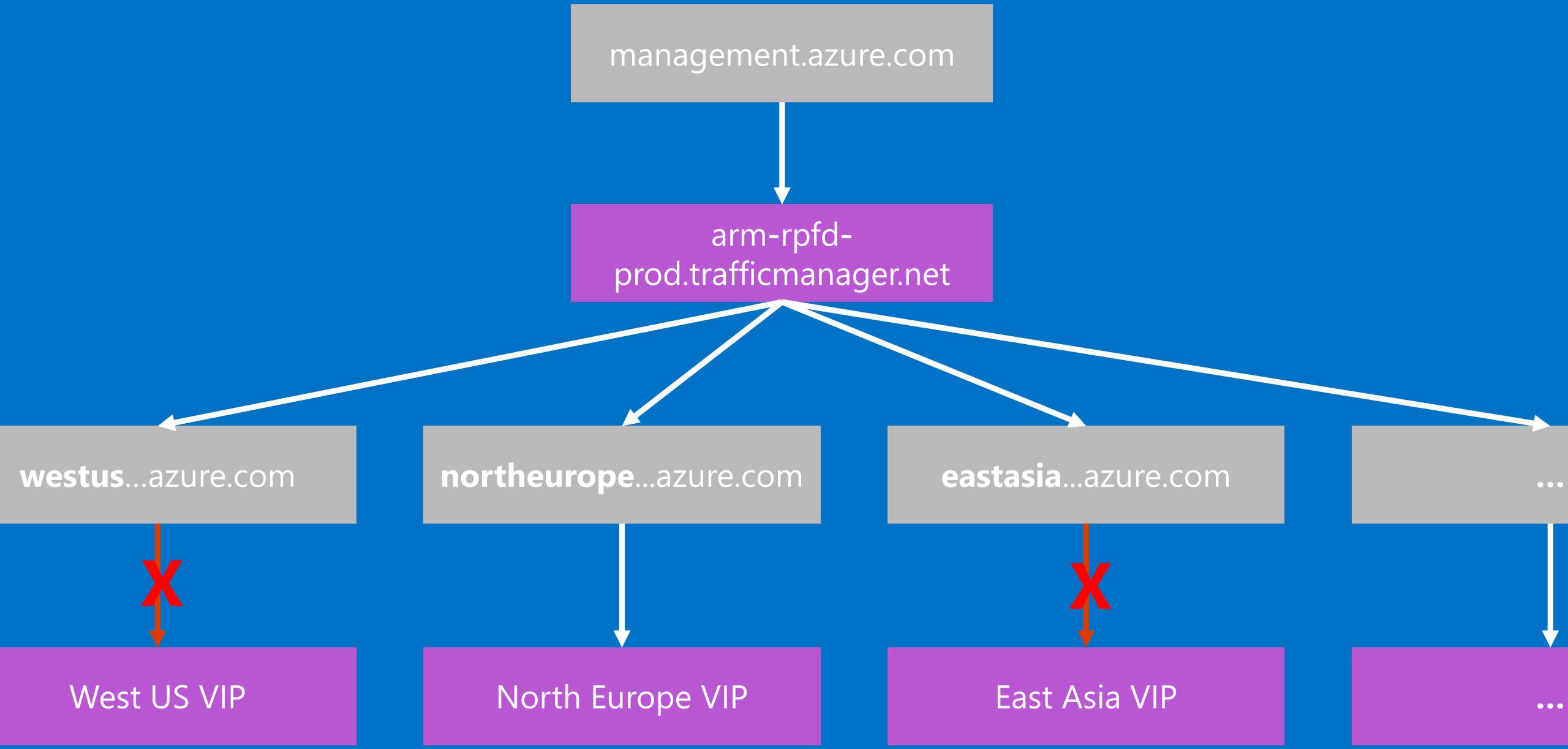
```
C:\>nslookup management.azure.com
Server:  co1-dns-05.redmond.corp.microsoft.com
Address:  10.222.118.22

Non-authoritative answer:
Name:    rpfd-prod-sn-01.cloudapp.net
Address:  40.124.46.15
Aliases:  management.azure.com
          arm-rpfd-prod.trafficmanager.net
          southcentralus.management.azure.com
```


management.azure.com
arm-rpfd-prod.trafficmanager.net
Traffic Manager has a probe every 30 seconds
Removes a regional CName if it sees an issue
X
X
westus...azure.com
northeurope...azure.com
eastasia...azure.com
...
West US VIP
North Europe VIP
East Asia VIP
...

Failing over a region

- False positives are okay – but false negatives are not
- Happens 1+ times a week; automatic based on health signals
- State management makes it so failovers are transparent / unnoticed by customers


management.azure.com
arm-rpfd-prod.trafficmanager.net
westus...azure.com
northeurope...azure.com
eastasia...azure.com
...
West US VIP
North Europe VIP
East Asia VIP
...

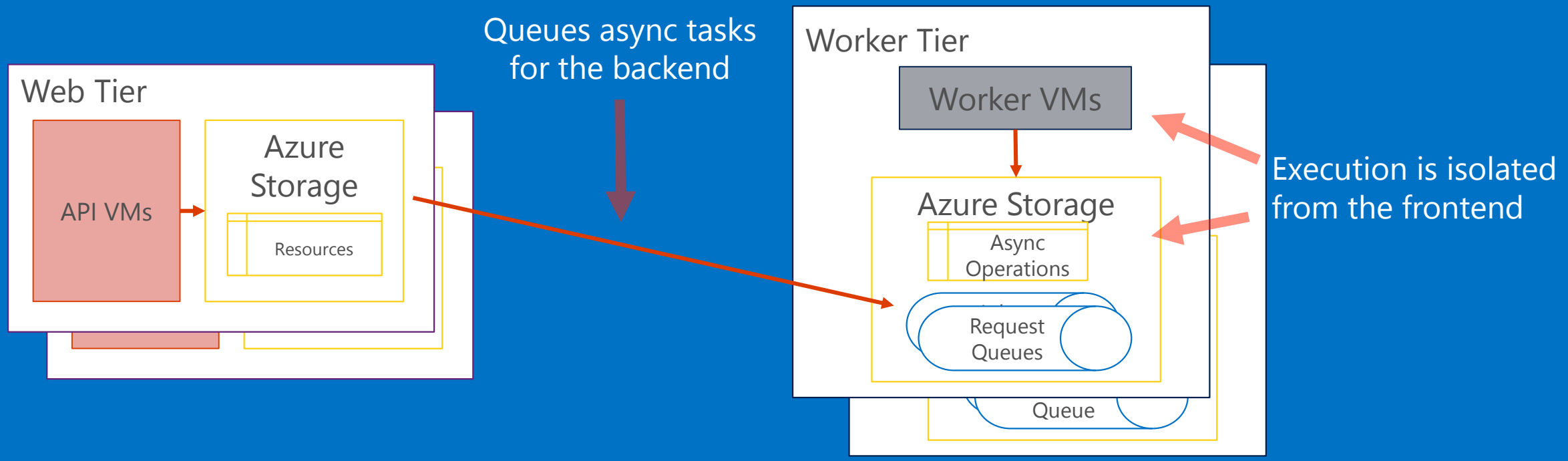
Failing over a VIP

- Remember: everything can go wrong
- VIP can get broken / deleted / stuck (due to hardware or platform issues)
- Regional hostnames can be mapped to a new VIP transparently to recover

Application Tier

High level architecture

- Stateless web tier for processing API requests
- Stateless worker tier for async tasks



Web tier

- Web tier sits behind a standard Azure SLB (same capabilities as ELB from AWS)
- Basic round-robin algorithm for forwarding requests
- Uses public MSFT stack: C#, IIS and Web API

- Non-persistent disks (ephemeral only) for less dependencies

Web tier

- SLB has a “probe” for each VM to test health; automatically will remove a node if unhealthy

- Try to remove unhealthy node(s) before DNS removes the whole region

Worker tier

- Worker tier does all heavy lifting (e.g. creating a VM cluster) asynchronously from API requests
- C# and standard Azure PaaS (Cloud Services)

- Non-persistent disks (ephemeral only) for less dependencies

- Worker tier has no ports open (only pulls tasks from queue)

Worker tier

- If overloaded due to requests –
  - most management operations will still succeed (since web tier will not be impacted)
  - async operations will run slower, but eventually complete (grey failure for customers)

State Management

State Management

- Needed to find a way to build an AP system that has huge scale
- No out-of-the-box storage service that spans 15+ regions reliably & performantly
- Evaluated SQL, Cassandra, Azure Storage, other internal MSFT stores

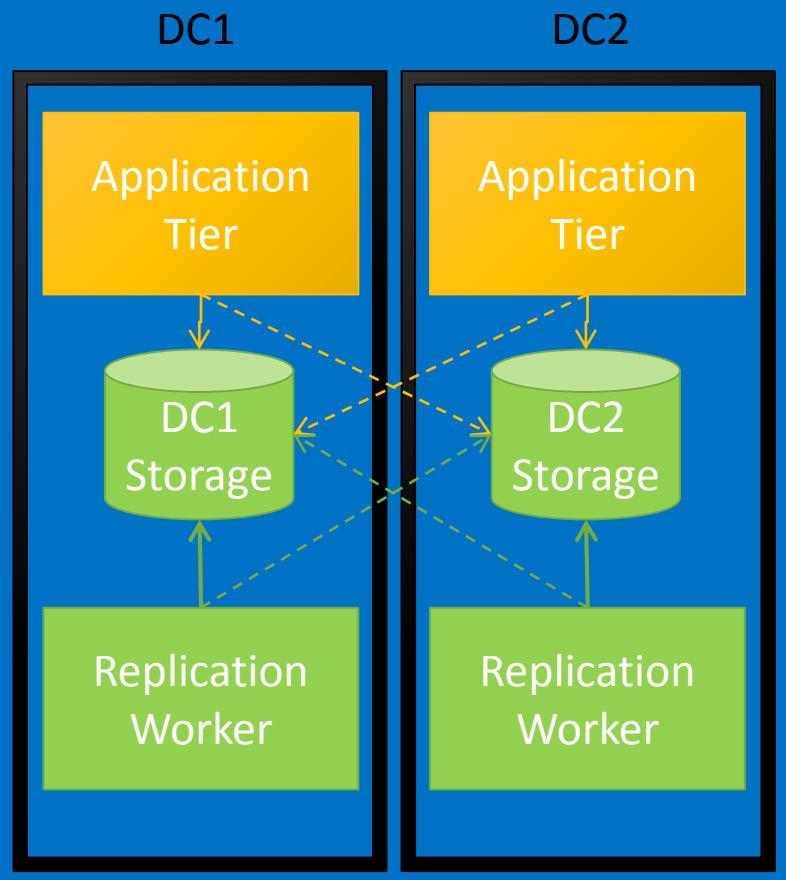
Selecting our solution

- Decided to take an off-the-shelf regional key/value store (Azure Table, REST based) and add a layer on top

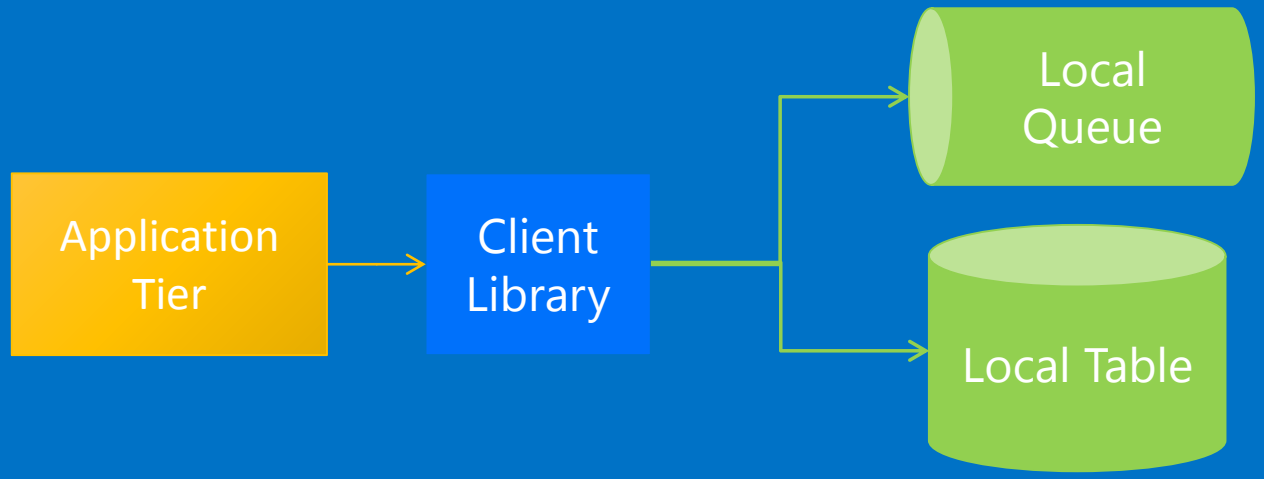
- If our layer fails – we would “fall back” to a proven, public data store

Active-Active-Active Tables

- **Asynchronously** replicates data across tables
- Multi-master system
- System is AP (no consistency)

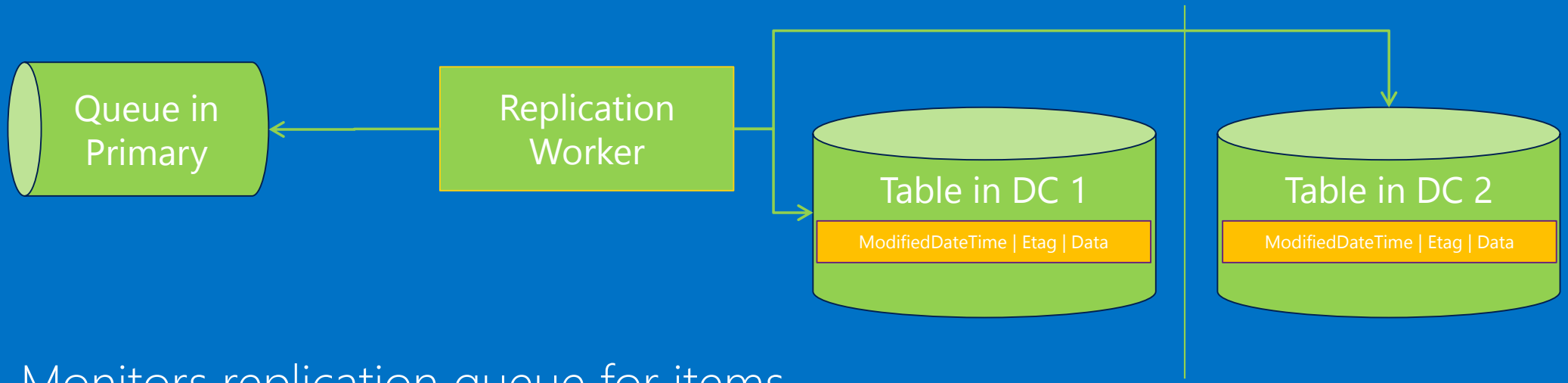


How does a write work?



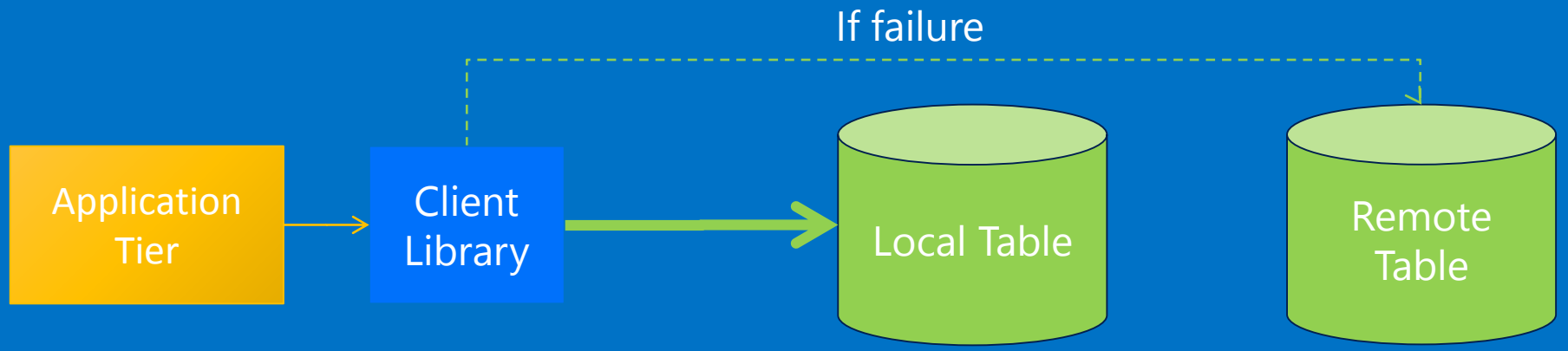
1. add an item to replication queue (10 minute visibility delay)
2. save data to local table
3. add an item to replication queue (no visibility delay)

How does replication work?



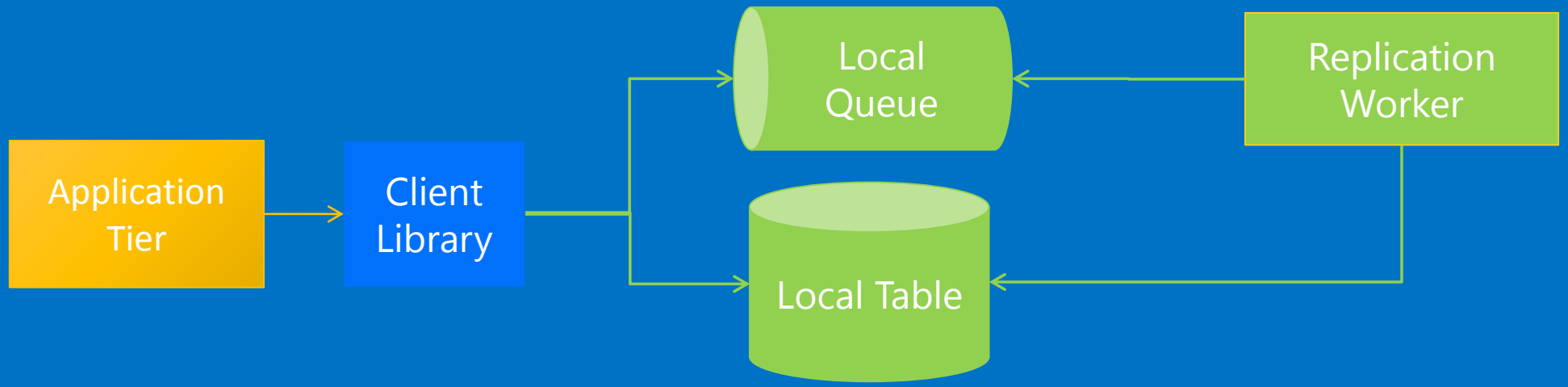
1. Monitors replication queue for items
2. Replication item just has PK of the record – indicates dirtiness
3. Replication worker reads the record for that PK in all tables in the ring
4. Replication worker chooses the "winner" (based on application details, timestamps, and ordering fields)
5. Replicates "winner" to all storage accounts using an ETag
6. If #5 fails for any storage account, or #3 was a partial read, requeue the item with exponential retry
7. Else, delete the item from the queue

How does read work?



- Read data from local table and only use remote tables if it fails
- Originally had a “read repair” concept but just hurt perf and never revealed issues

Important Notes



- No service between application tier and Azure Table
- If replication service is down / broken, saves continue to work (but replication items grow in the queue)

Deletions

- A common problem with a system like this is deletes – e.g. once a record is purged, an "older" update can cause it to be resurrected

- Solved by
  - using soft deletes (adding a "deleted time" but not actually deleting it)
  - large grace period before hard deletion
  - "two pass" purges

How do we hide eventual consistency from a customer?

- Latency for full replication is <5 seconds in 99th percentile **(production today)**

- Saving + refreshing a page would hit this if we did not have protections

Giving a customer a single view

- If a customer is not physically moving, they will hit the same regional endpoint (and therefore the same storage account).. which makes their replica consistent

- If two customers on opposite ends of the globe are looking at the same assets, “suspicious” requests trigger a cross-region scan

Data Snapshotting (Durability)

Each table has regular snapshots taken to allow for a point in time restore of the data (e.g. corruption)

Snapshots are stored in alternate storage accounts in disparate locations

Snapshots are maintained for a period of time (90 days) before being deleted

Daily Data Repair

Performs consistency check between storage accounts

Fixes any inconsistencies or issues on demand

High throughput, easily partitionable task (e.g. nodetool repair in Cassandra)

Used to bring up new regions (table starts empty)

Recovering a downed region

60 Minutes * 20,000 Transactions / Sec = 72 Million writes

Need to run at a fraction of overall capacity (e.g. 10%<) and support “bursting” to recover even faster

Adoption of the storage capabilities

- Spreading through Microsoft – 12+ products use it today

- Open sourcing the client library + replication service

Deployments

How do deployments work?

- "One click" to start, all engineers have permissions
- Automatic rollout strategy that does everything for the engineer over two days

What is the deployment strategy?

1. Deploys to “canary” region (not exposed to customers, but has partner tests / monitoring)
2. Waits a few hours
3. Deploy low traffic regions public off-peak
4. Waits a few hours; go to #3 if regions remain

What safeguards are in place?

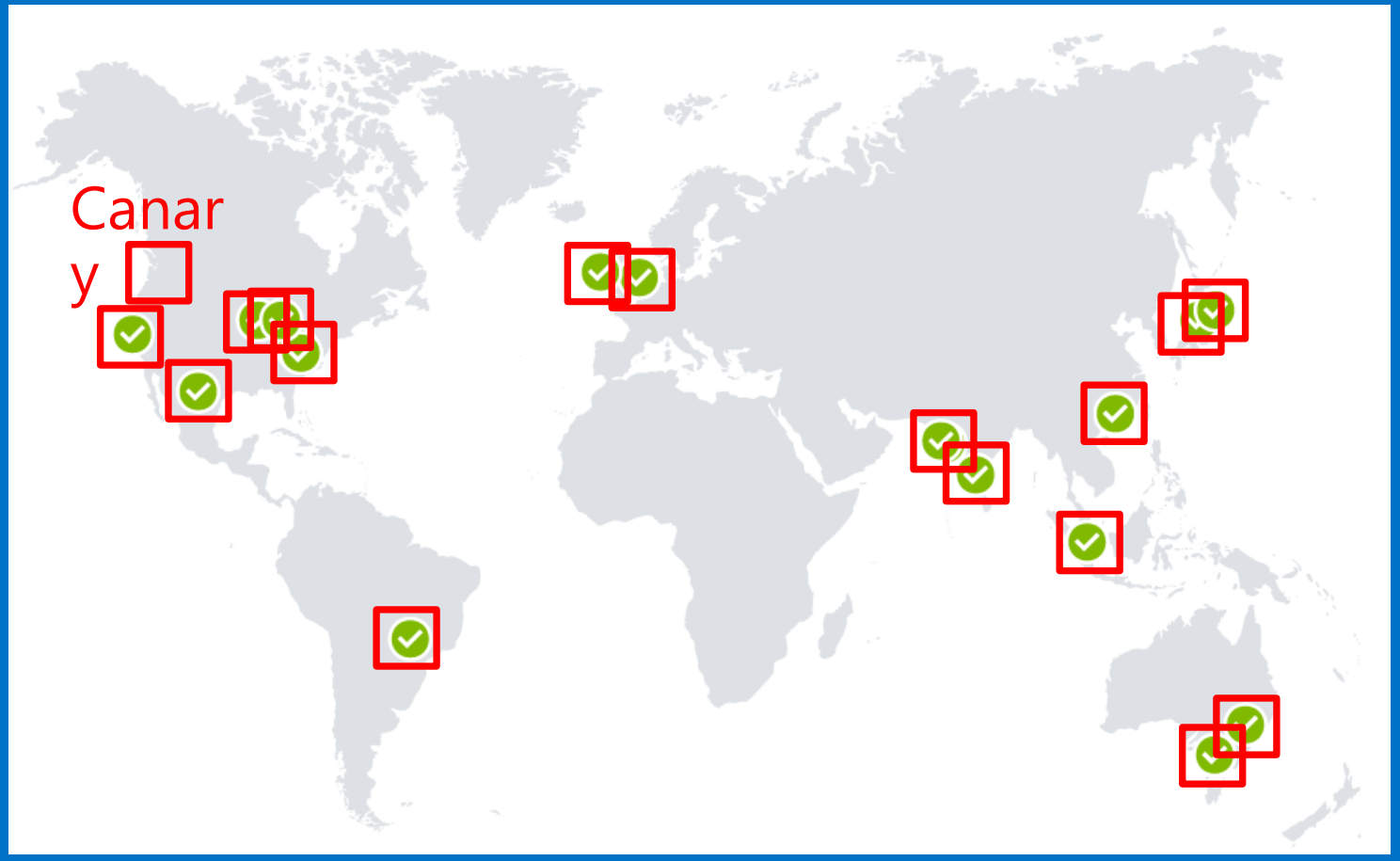
- Pauses outside core business hours (10 AM – 4 PM)
- Spans two days, including peak traffic of a major region (to catch perf / load issues)
- If monitoring fails, suspends deployment automatically
- Anti co-location: never deploys all the regions for a continent at once

Real world example

0. March 1, 12 PM (PST)
1. March 2, 10 AM
2. March 2, 1 PM
3. March 2, 4 PM

4. March 3, 10 AM (PST)
5. March 3, 1 PM

Done!



Internal Routing

Routed to an internal microservice

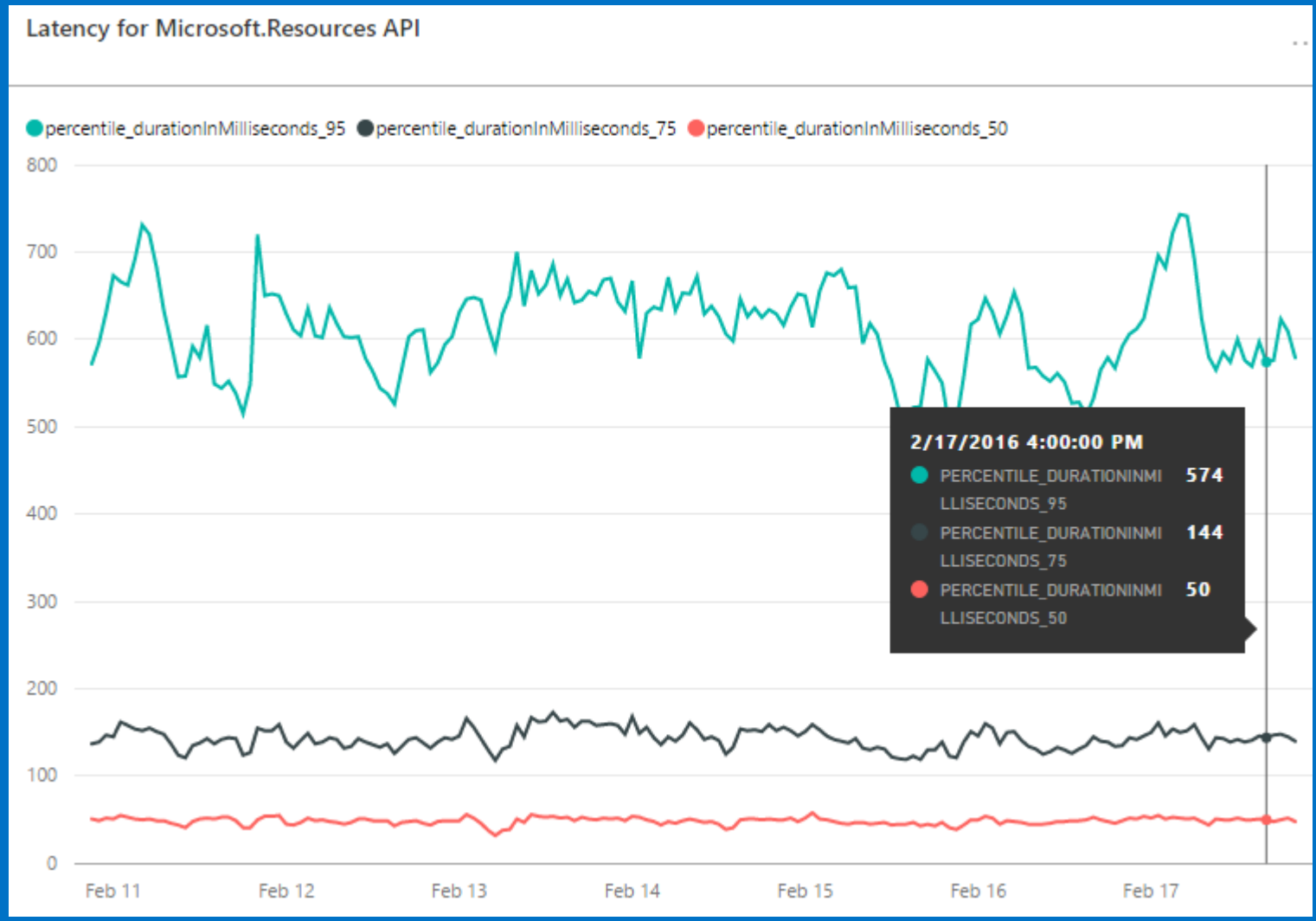
- Frontdoor inspects the request (e.g. is it provisioning a Virtual Machine or Load Balancer? what region is the user requesting?)
- Matches several items on the request to identify the right microservice endpoint to call
- Done via a declarative manifest published to frontdoor

What were the results?

Substantial performance improvements

50%+ for primary customer APIs

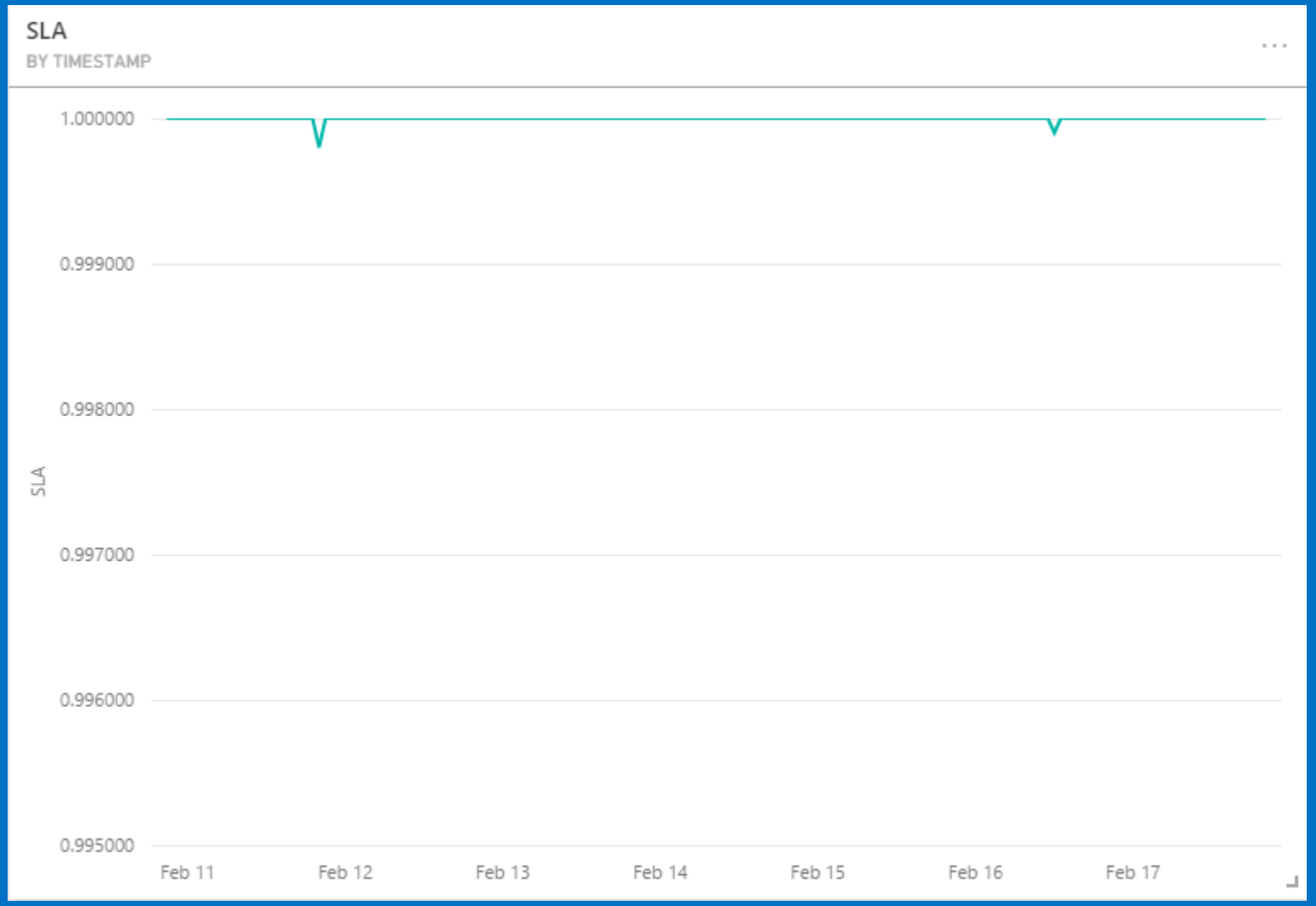
Extremely consistent



Incredible availability

No global outages, period (2 years)

SLA is met hourly (and 5+ 9's weekly)



Feature velocity

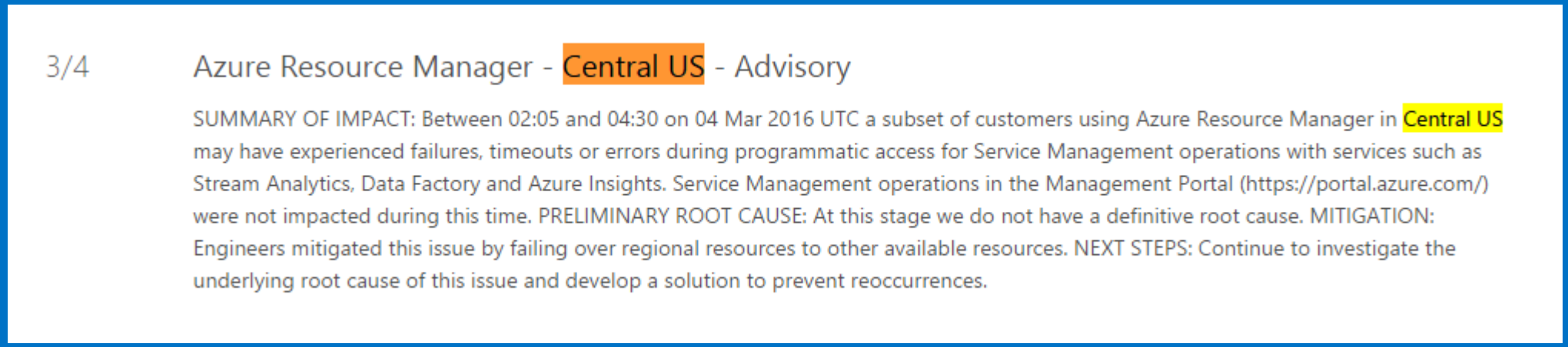
- 300%+ increase in code velocity – new frontdoor features released several times each week (and underlying microservices ship independently)

- Anyone on the team can push changes confidently and with one click – rollout is fully automated

What's left?

Automatically detect “soft” failures

DNS & node health checks detect “hard” failures (e.g. networking down), but bad for “soft” failures



Protect against “poisonous” clients

- Particular customer or partner may have a call pattern that trips up a bug / perf problem
- Automatic failover will kick in and transfer the “poisonous” traffic region-by-region
- Quarantine region + traffic until fix can go out

Automatic threshold / anomaly detection

- All performance / failure ranges are manually specified by an engineer (e.g. if latency is > 1 [s], mark the region as failed)

- Instead, ask engineers to identify key metrics and have the system detect anomalies over time / on new builds

Questions? (we're hiring)

@clamanna
chlama@microsoft.com