goto; copenhagen

GOTO Copenhagen 2017
Conference Oct. 1-3, 2017

High Availability Distributed (Micro-)services

Clemens Vasters – Microsoft Azure
@clemensv

Follow us @gotocph

goto; copenhagen

GOTO Copenhagen 2017

Conference Oct. 1-3, 2017

ice

Click ‘Rate Session’ to rate session and ask questions.

Follow us @gotocph

Microsoft Azure services I work(-ed) on….

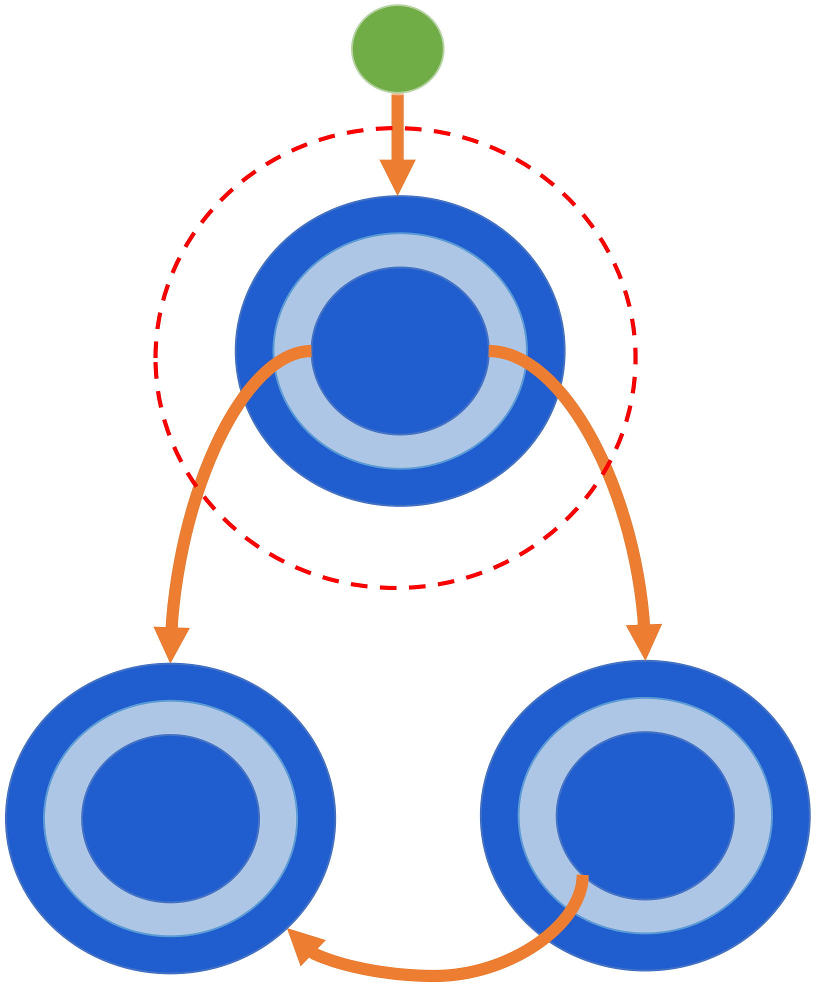
Notification Hubs	**Service Bus**	**Event Hubs**	**Event Grid**	**IoT Hub**	**Relay**
Mobile push notifications	Cloud messaging	Telemetry stream ingestion	Event distribution	IoT messaging and manage-ment	Discovery, Firewall/NAT Traversal

Azure Messaging by the numbers

5.1 Trillion Requests per week in Event Hubs	8,432,540 Requests per second average 24/7	99.9984% Success Rate	50ms Average Event Hubs send latency
>28 PB Monthly data volume	1.8 Million Message Queues and Topics in production	>100,000 Daily active Service Bus Namespaces	695 Billion Message operations on Azure Service Bus Messaging per month

A “Service” is software that …

- … is responsible for holding, processing, and/or distributing particular kinds of information within the scope of a system
- … can be built, deployed, and run independently, meeting defined operational objectives
- … communicates with consumers and other services, presenting information using conventions and/or contract assurances
- … protects itself against unwanted access, and its information against loss
- … handles failure conditions such that failures cannot lead to information corruption



Services: Autonomous Entities

- Defining property of services is that they’re *Autonomous*
  - A service owns all of the state it immediately depends on and manages
  - A service owns its communication contract
  - A service can be changed, redeployed, and/or completely replaced
  - A service has a well-known set of communication paths
- Services shall have no shared state with others
  - Don't depend on or assume any common data store
  - Don't depend on any shared in-memory state
- No sideline communications between services
  - No opaque side-effects
  - All communication is explicit
- **Autonomy is about agility and cross-org collaboration**

Interdependencies

- An autonomous service owns its own uptime
  - If a downstream dependency service is unavailable, it may be acceptable to partially degrade capability, but it’s not acceptable to go down blaming others
  - Any critical downstream dependencies need to be highly available, with provisions for disaster recovery.
  - A service can rely on a highly-available messaging middleware layer as a gateway to allow for variable load or servicing needs
- An autonomous service honors its contract
  - Version N honors the contract of Version N-1. Contracts are assurances.
  - Deprecation of a contract breaks dependents; have a clear policy

Operational Assurances

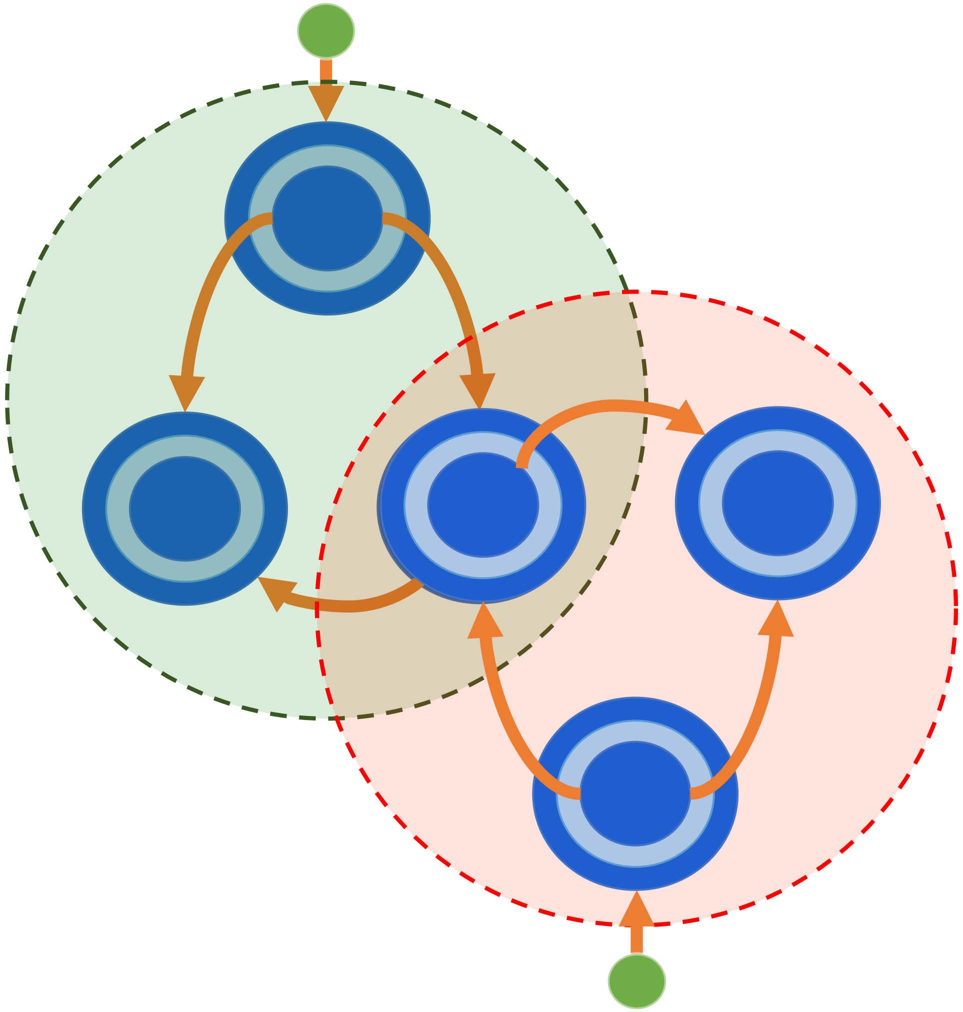
- Service owners aim to meet operational objectives so that they can provide operational assurances:
- What level objective achievement can and does the service owner commercially commit to?
  - Example: Operational objective 99.99% availability/week (10 minutes max downtime) might turn into assurance 99.95% (50 minutes max downtime)
  - Latency? Throughput? Data Loss? Disaster/Failure Recovery Time?
- What is the support lifecycle commitment for APIs and contracts?
  - How many versions? Minimum deprecation notice?

The modern notion of “Service” is not about code artifact counts or sizes or technology choices.

It’s about ownership.

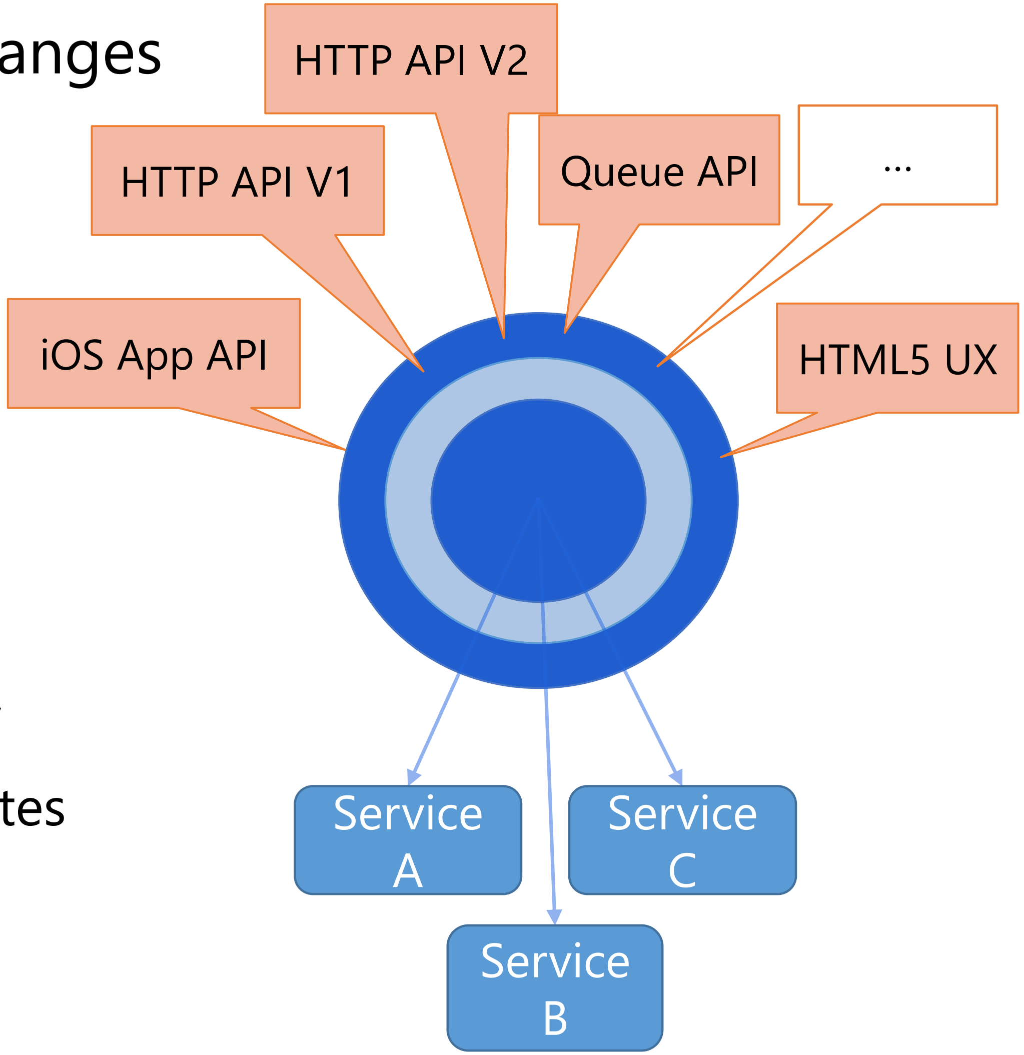
System

- A system is a federation of services and systems, aiming to provide a composite solution for a well-defined scope.
- The solution scope may be motivated by business, technology, policy, law, culture, or other criteria
- A system may appear and act as a service towards other parties.
- Systems may share services
- Consumers may interact with multiple systems



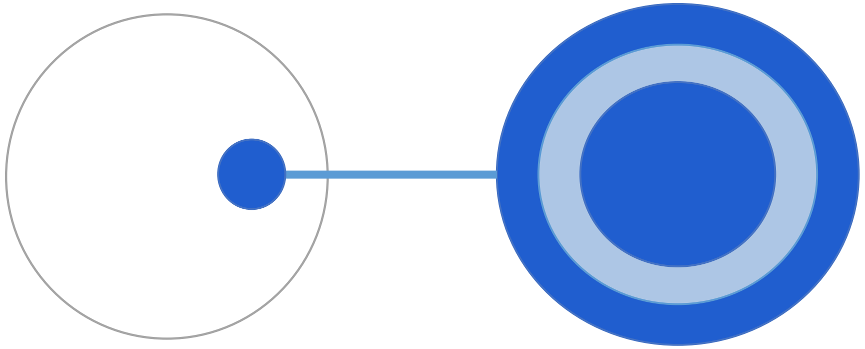
Rationale for Layers

- Key rationale for layers: Resilience against changes in ambient contracts.
- Communication and Presentation Layers
  - Lots of changes, fairly frequently
    - New UX methods and layouts, new assets
    - New contracts and schemas
    - New protocols
  - Can have multiple concurrent interfaces
  - Each change has low impact, but work adds up
- Resource Access Layers
  - Fewer changes, rather infrequently
    - Downstream dependency services make compatibility assurances
  - Sometimes massive impact, often wholesale rewrites
- Goal is for core logic to be resilient against interface changes



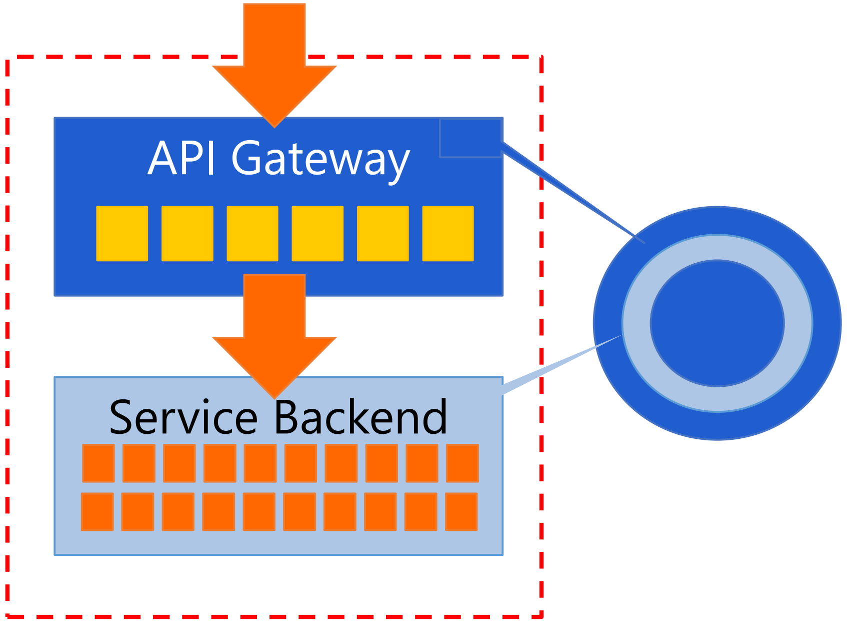
“Fiefdoms and Emissaries”

- Term coined ~2002 by @PatHelland
- “Fiefdom”: Autonomous Service
- “Emissary”: Logic/Code
    - JavaScript on Web Pages
    - Client SDKs
- „A service owns its contract“ can also manifest in it owning SDKs for all relevant platforms while keeping the wire contract private.
- We‘ll see more of this around „edge compute“

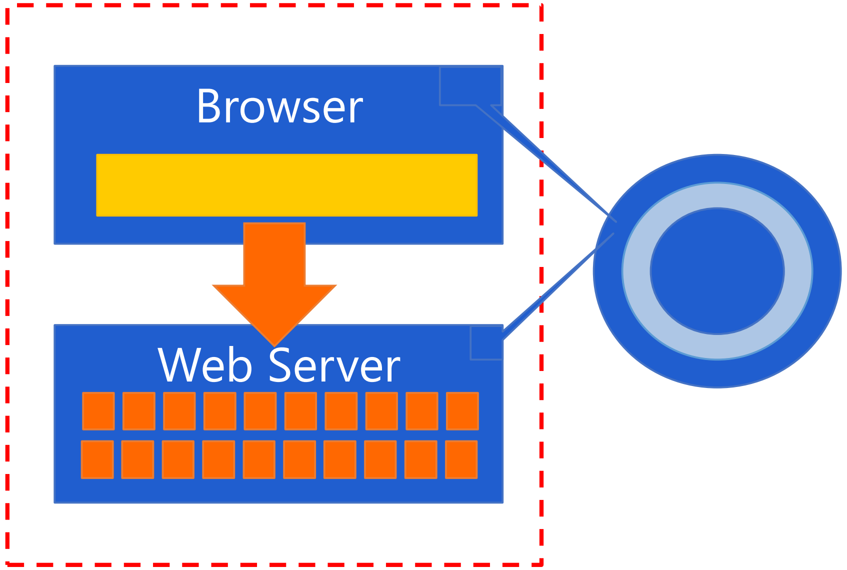


Tiers: Runtime Organization

- Tiers are about meeting operational objectives
  - Aspects of one service or even one layer may have different scalability and reliability goals
  - Resource governance (I/O, CPU, Memory) needs may differ between particular functions
  - UX tier will be more efficient and more adaptable with client-based rendering
- Tier boundary most often is a process boundary
  - On same machine, across machines
  - In same organization, across organizations
  - In trusted environment, across trust boundaries
- Tier boundaries often cut through layers
  - Cuts may separate "yours" and "theirs"
  - Ex: "Your" hosted web code and "their" browser
  - Ex: "Your" data access code and "their" database



2 tiers, 2 layers



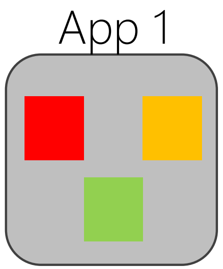
2 tiers, 1 layer

Services vs. Microservices

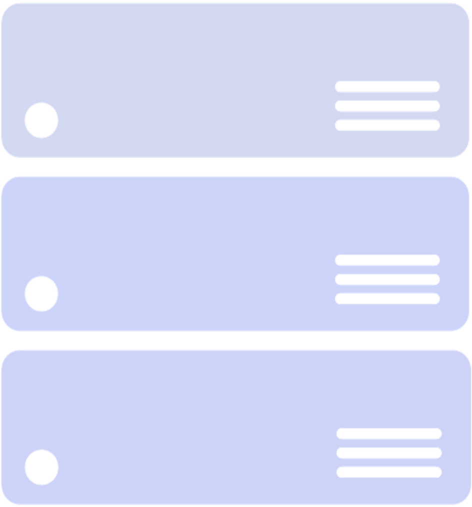
Running Tiers as Services

Runtime and Deployment Models

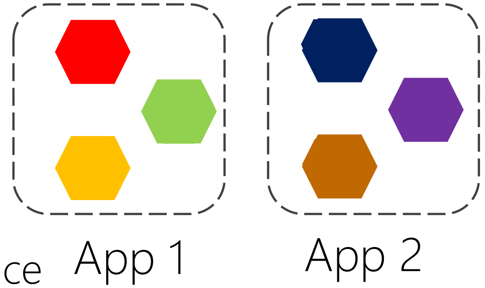
- A monolith app contains domain specific functionality and is normally divided by functional layers such as web, business and data

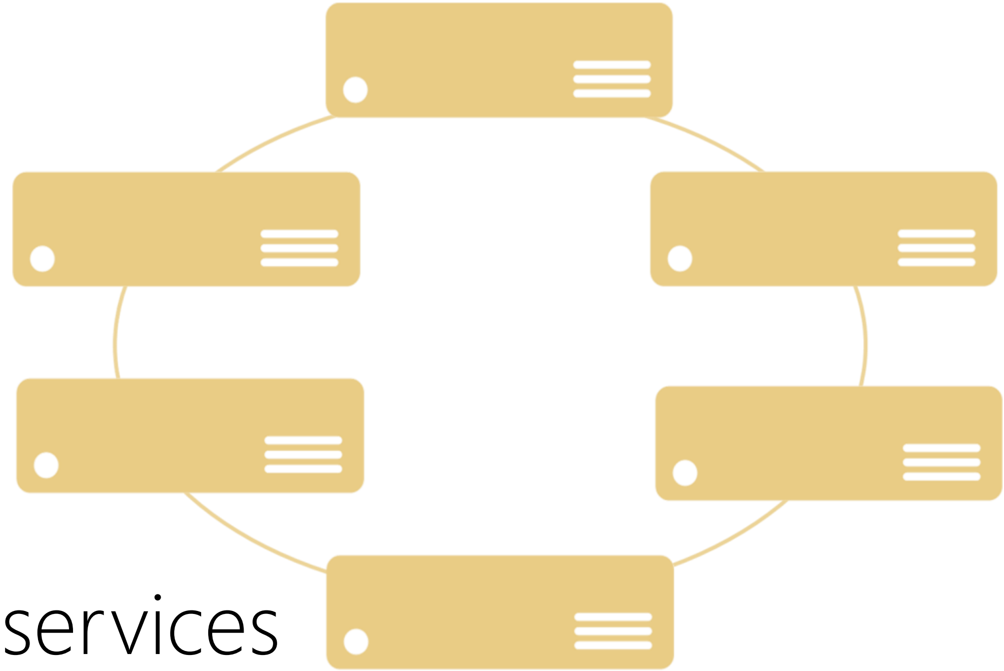


- Scales by cloning the app on multiple servers/VMs/Containers



- A microservice application separates functionality into separate smaller services.
- Scales out by deploying each service independently creating instances of these services across servers/VMs/containers



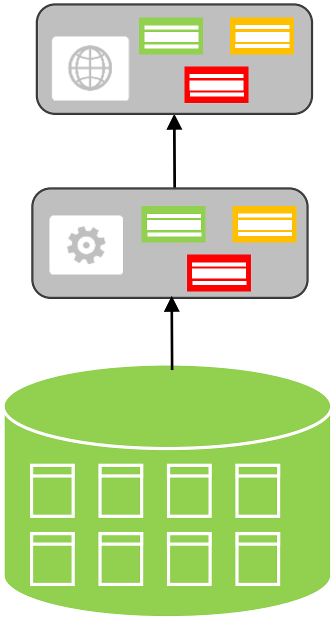


Monolith | Microservices

State Management

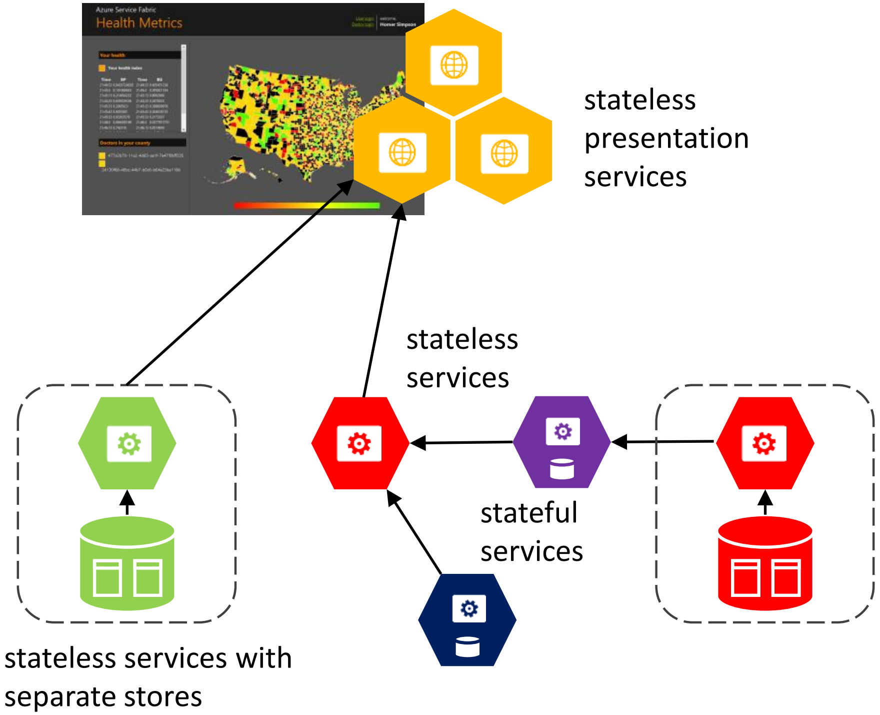
Monolithic approach

- Single monolithic database
- Tiers of specific technologies



Microservices approach

- Graph of interconnected microservices
- State typically scoped to the microservice
- Variety of technologies used



stateless presentation services

stateless services

stateful services

stateless services with separate stores

Microservice Architecture Benefits

Scale Independently

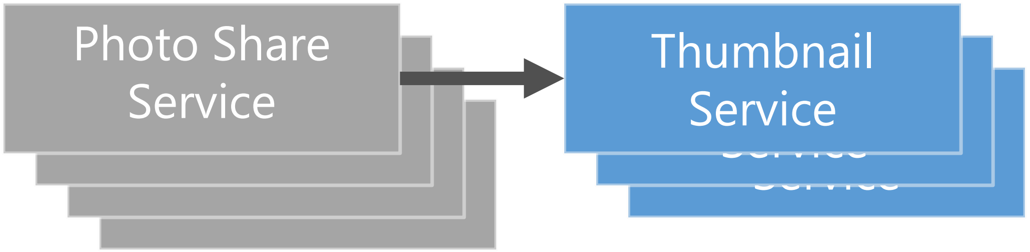


Photo Share Service

Thumbnail Service

Independent Deployments

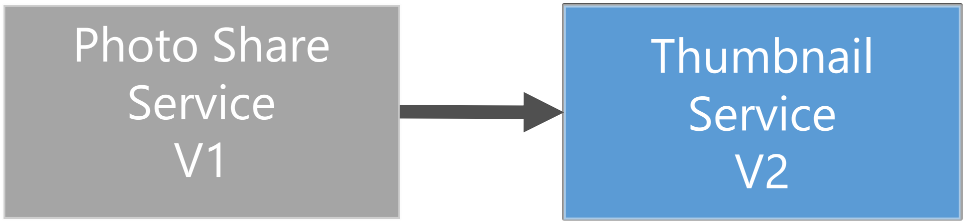


Photo Share Service V1

Thumbnail Service V2

Different Technology Stacks

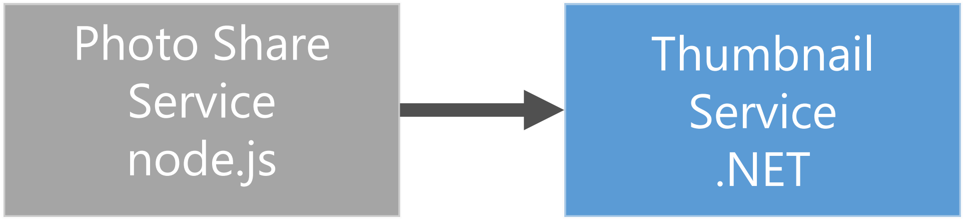


Photo Share Service node.js

Thumbnail Service .NET

Conflicting Dependencies

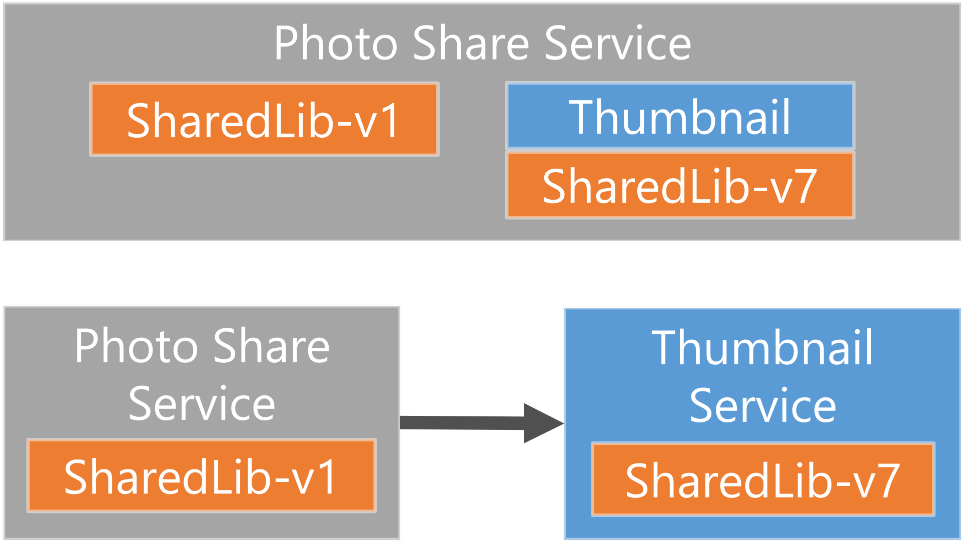


Photo Share Service

SharedLib-v1

Thumbnail

SharedLib-v7

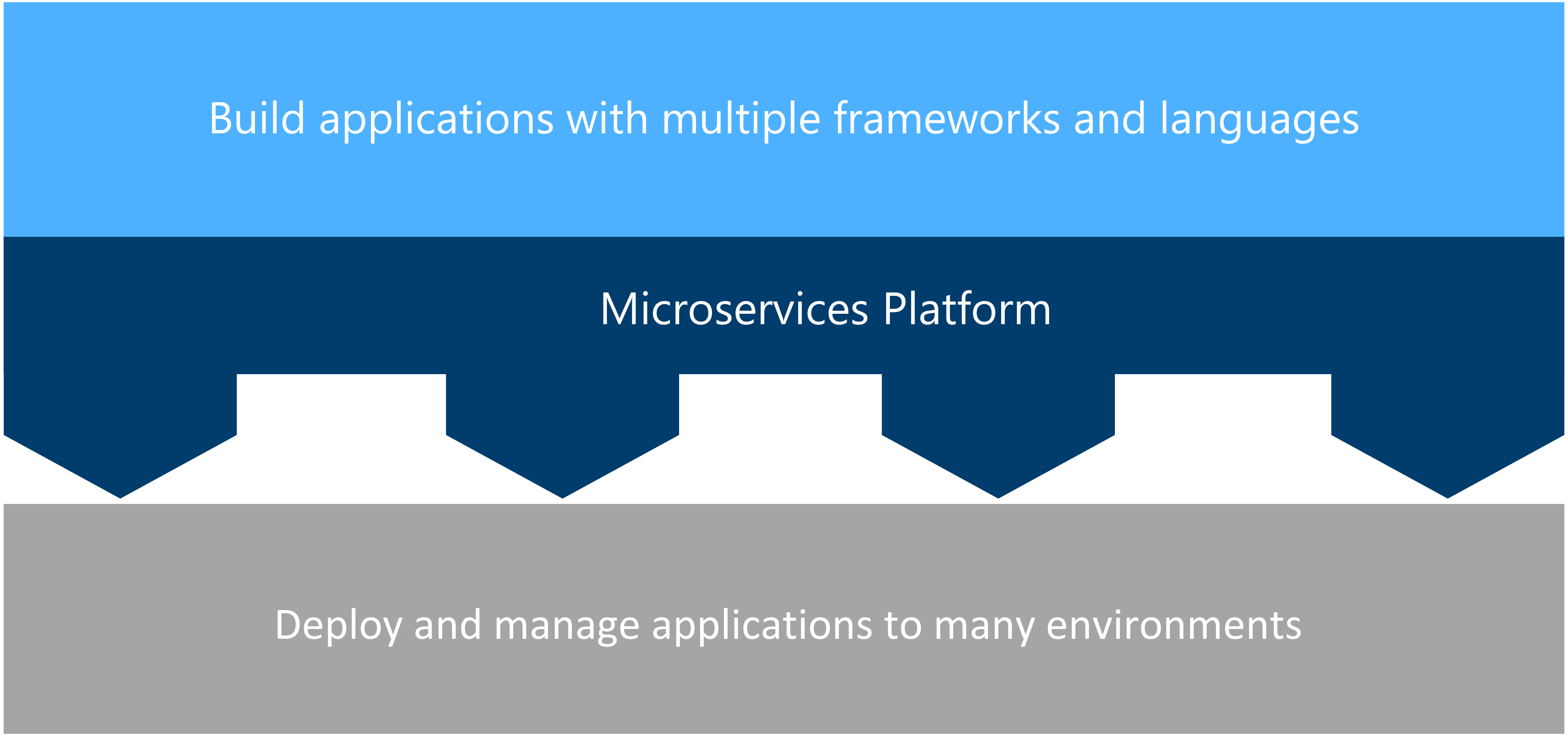
Photo Share Service

SharedLib-v1

Thumbnail Service

SharedLib-v7

Microservices Platform Requirements



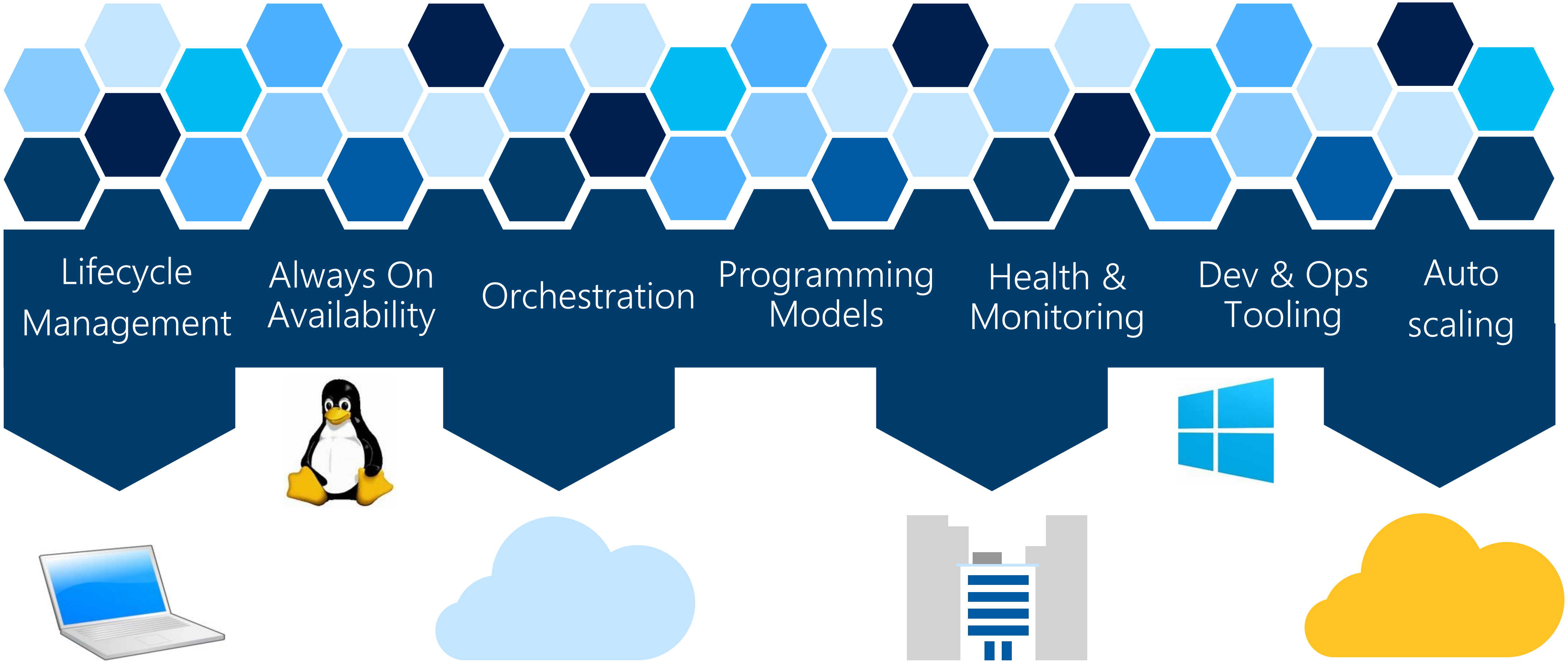
Build applications with multiple frameworks and languages

Microservices Platform

Deploy and manage applications to many environments

Azure Service Fabric

Any OS, Any Cloud



Lifecycle Management

Always On Availability

Orchestration

Programming Models

Health & Monitoring

Dev & Ops Tooling

Auto scaling

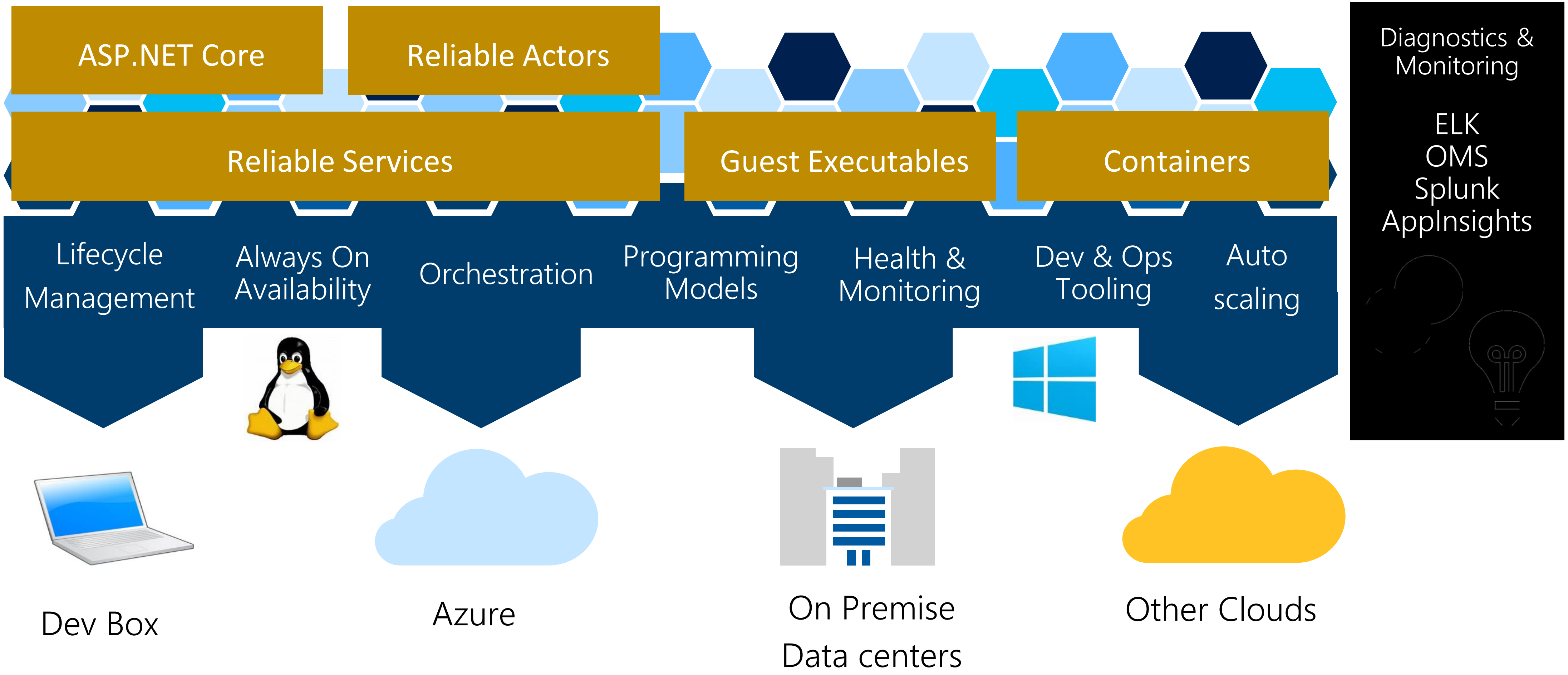
Dev Box

Azure

On Premise Data centers

Other Clouds

Service Fabric Programming Models & CI/CD



ASP.NET Core

Reliable Actors

Reliable Services

Guest Executables

Containers

Lifecycle Management

Always On Availability

Orchestration

Programming Models

Health & Monitoring

Dev & Ops Tooling

Auto scaling

Diagnostics & Monitoring

ELK
OMS
Splunk
AppInsights

Dev Box

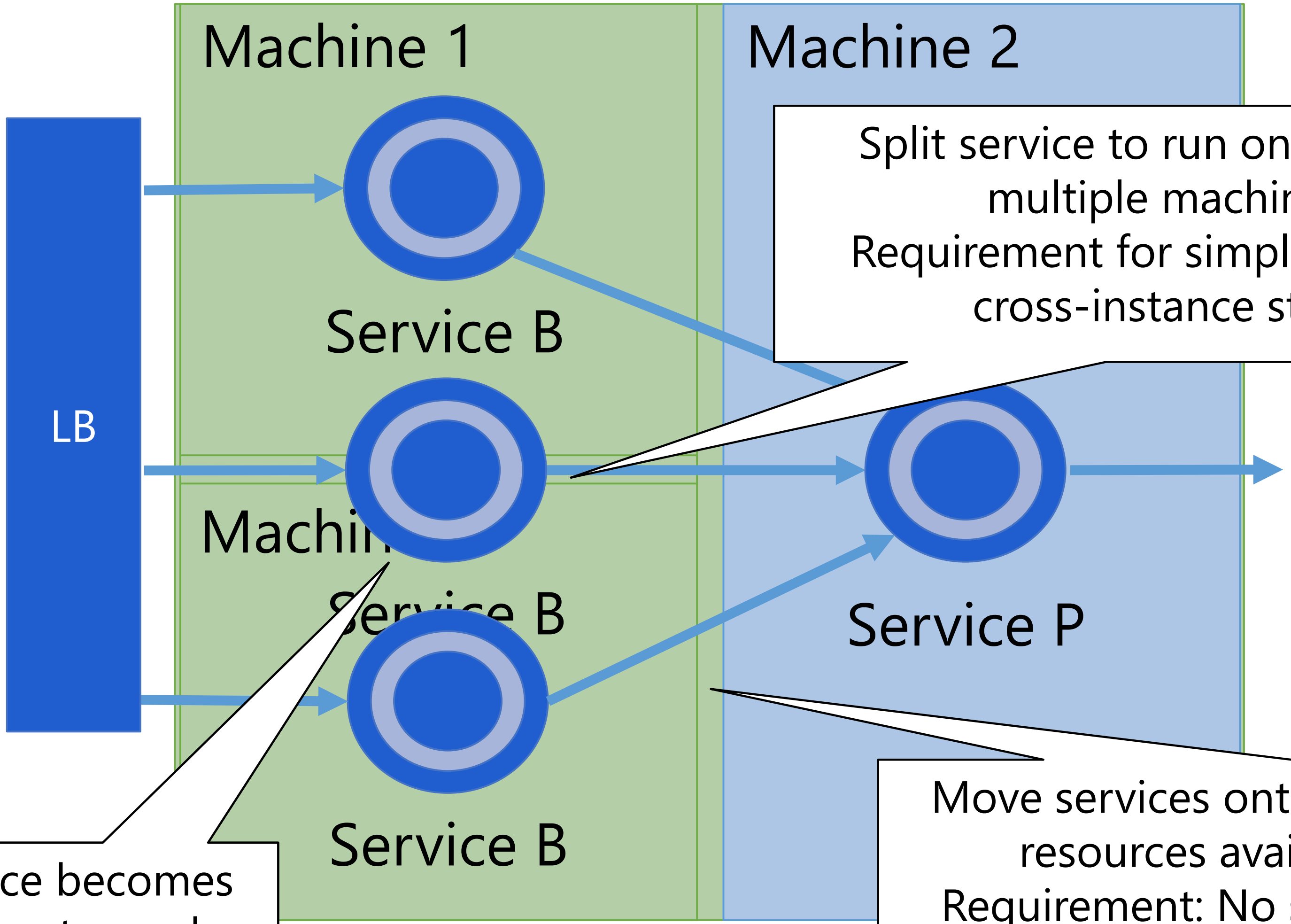
Azure

On Premise Data centers

Other Clouds

Scale and Reliability

Clustering

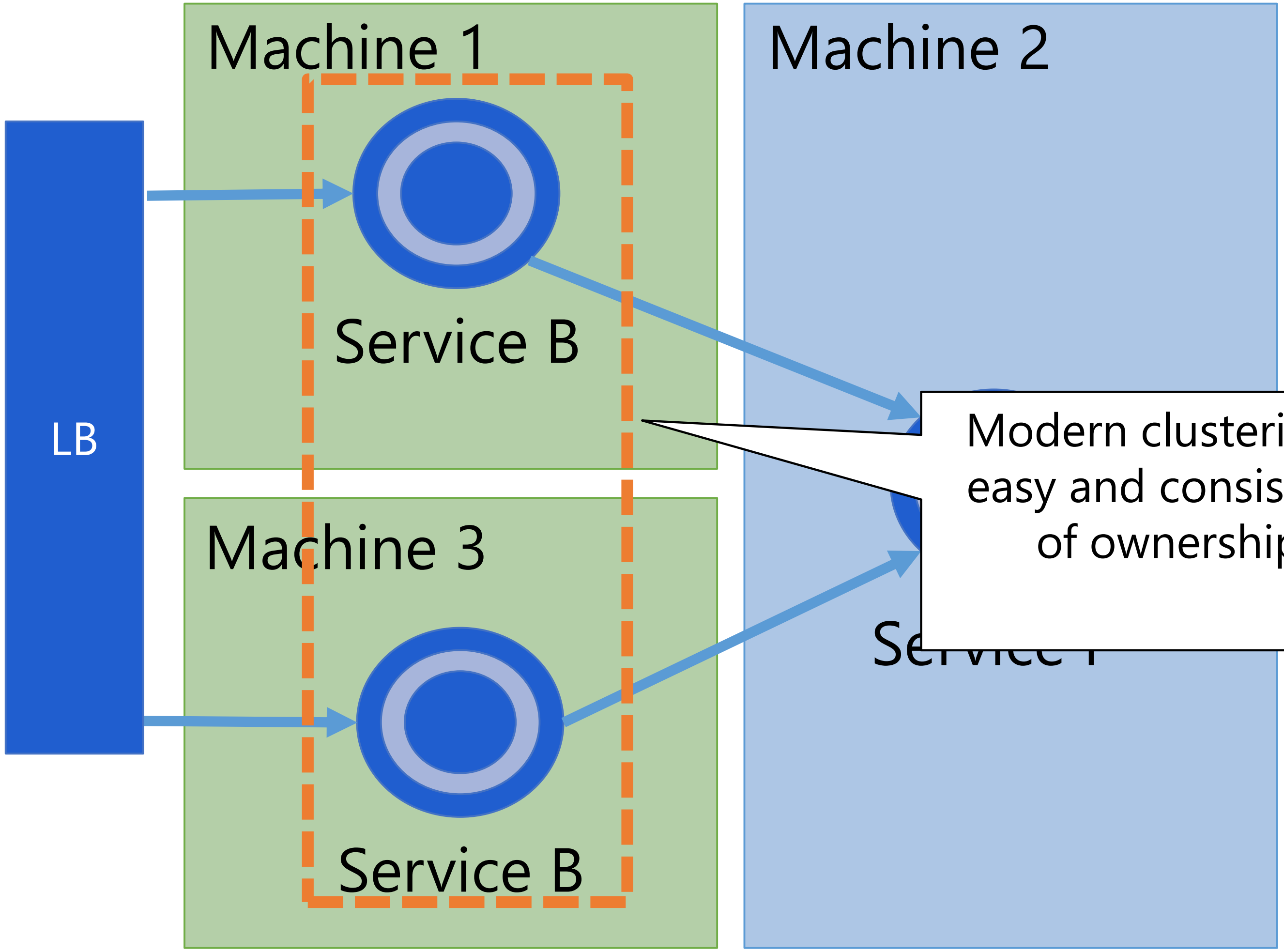


Split service to run on cluster of multiple machines.
Requirement for simple case: No cross-instance state.

Service becomes over-stressed

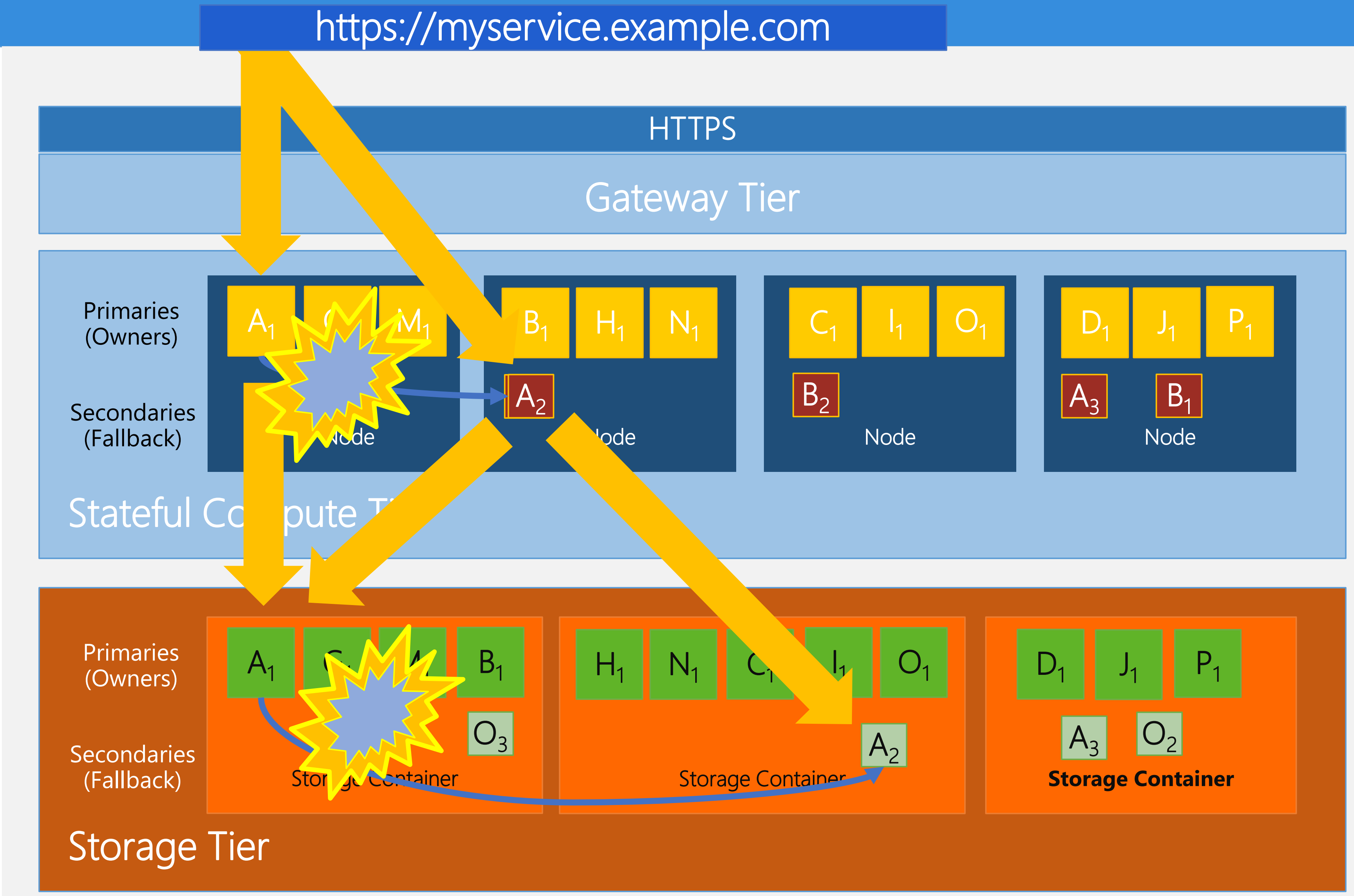
Move services onto multiple machines. More resources available to both services.
Requirement: No shared cross-service state.

Clustering



Modern clustering infrastructure can allow for easy and consistent state-sharing and failover of ownership for aspects of partitioned workloads

Multi-Node Failover Clustering



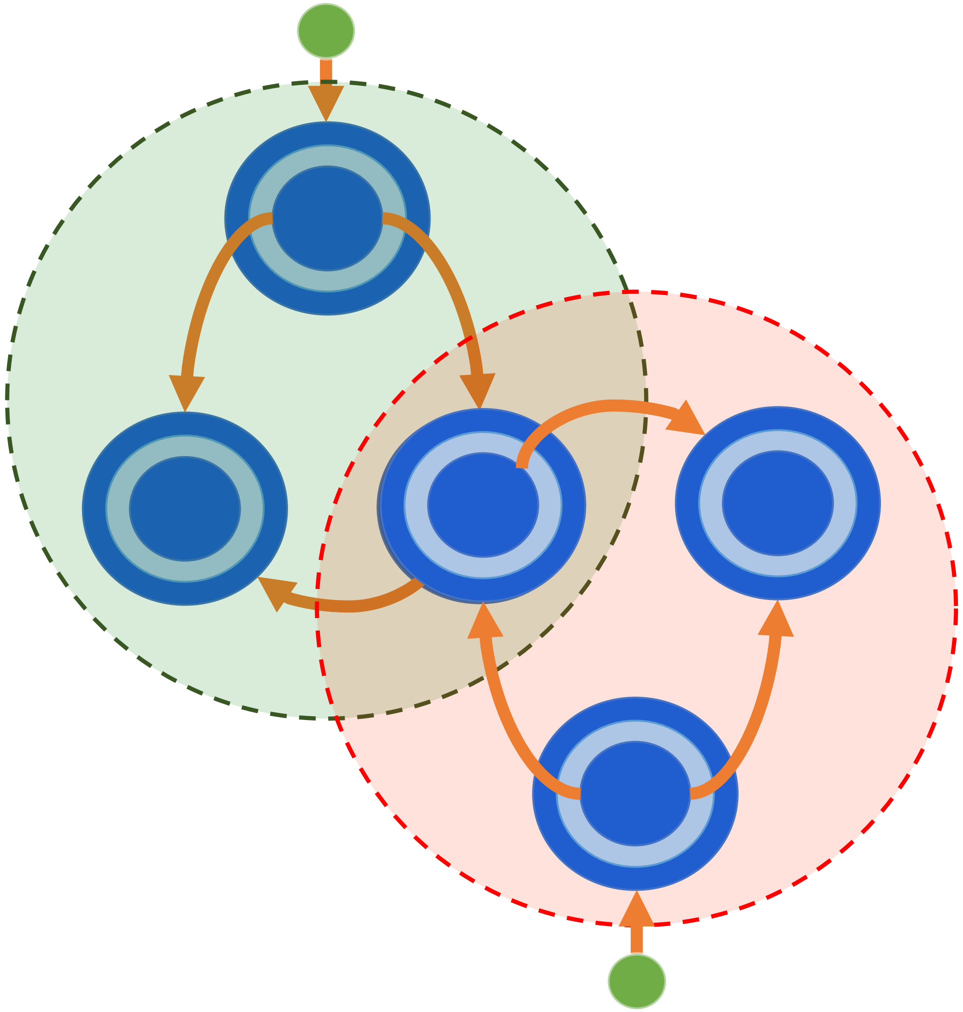
Failure of any node – in gateway, compute, or storage – leads to an automated "failover" to one of at least two secondaries.

Secondaries are continuously updated with the all information required to instantly take ownership when needed.

RPC, Messaging, and Eventing

Communication

- “REST” is great for interactively accumulating and acting on state from multiple sources.
  - Let‘s not pretend all clients are like that – there‘s a lot more
- HTTP and RPC are great to obtain immediate answers.
  - The longer it takes to generate the answer, the more brittle the model becomes



Command

Report

Transfer

Notification

Query

Measurement

Job

Assignment

Handover

Update

Request

Trace

Command

Transfer

Query

Handover

Job

Assignment

Update

Request

Report

Notification

Measurement

Trace

Intents

Facts

Messaging

Intents

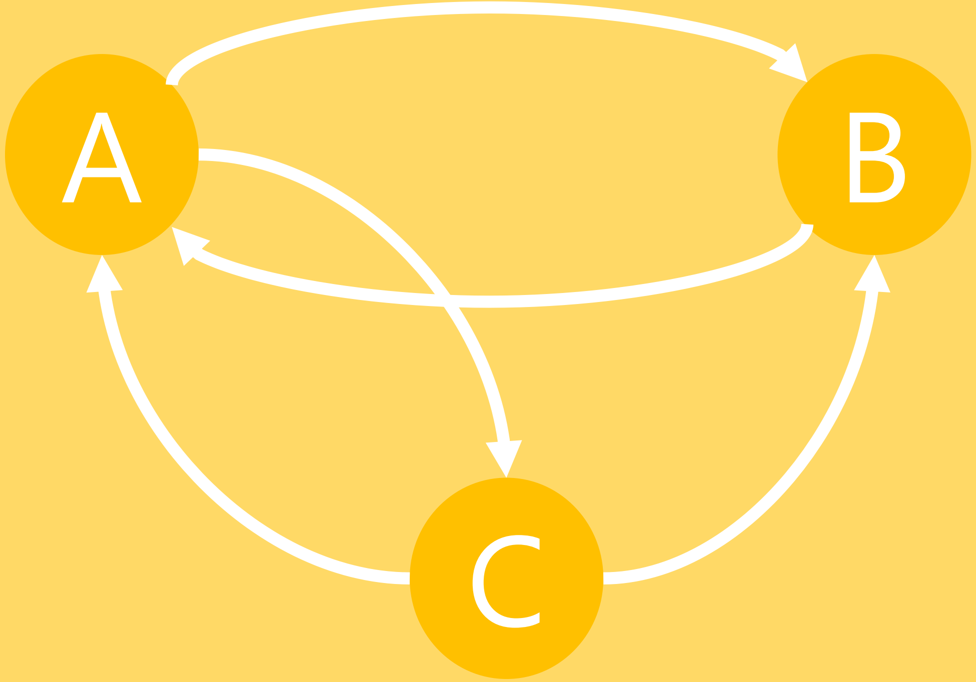
Expectations
Conversations
Contracts
Control Transfer
Value Transfer

Eventing

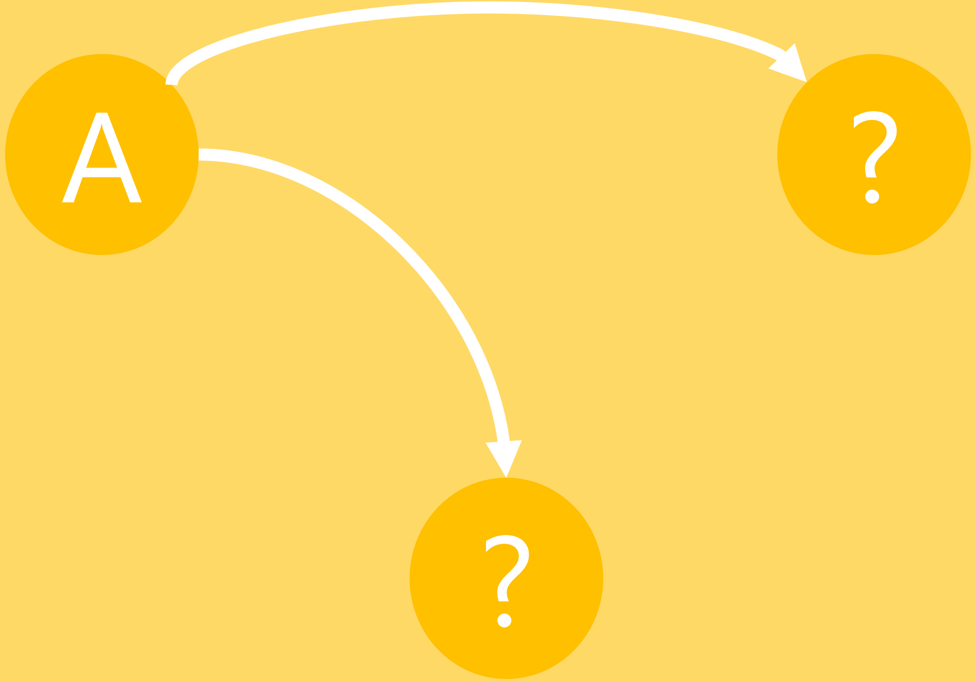
Facts

History
Context
Order
Schema

Messaging



Eventing



Events

Discrete

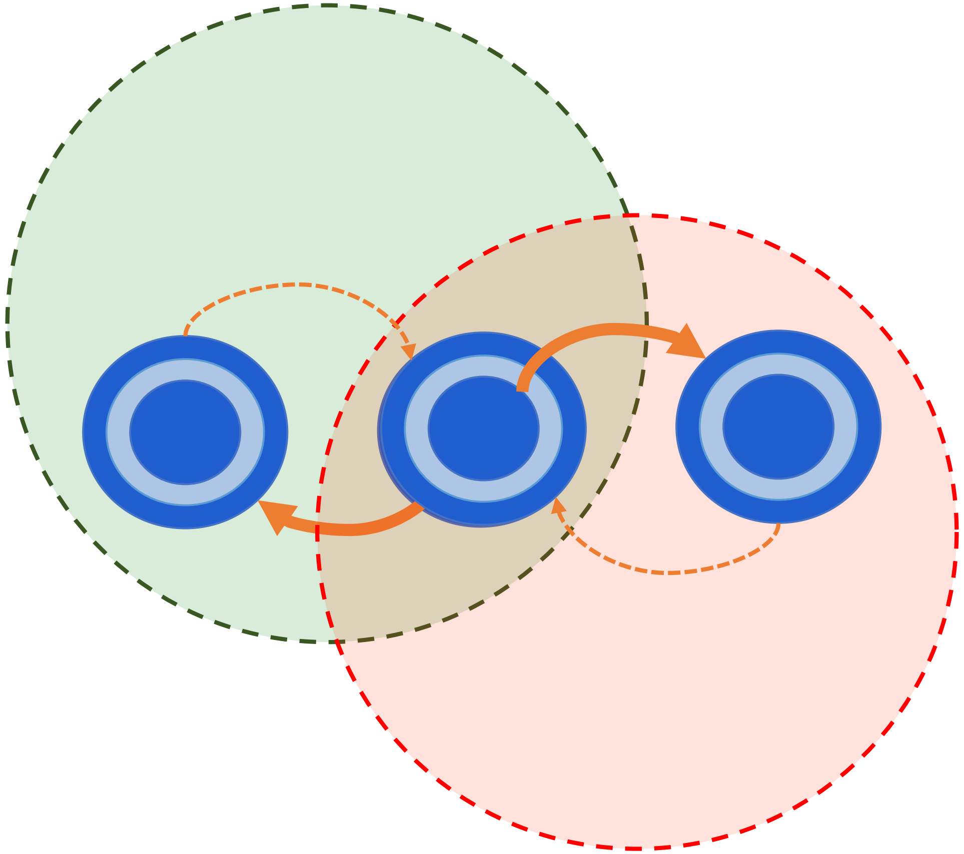
Independent
Report State Change
Actionable

Series

Time Ordered
Context Partitioned
Report Condition
Analyzable

Discrete Events are an Extensibility Enabler

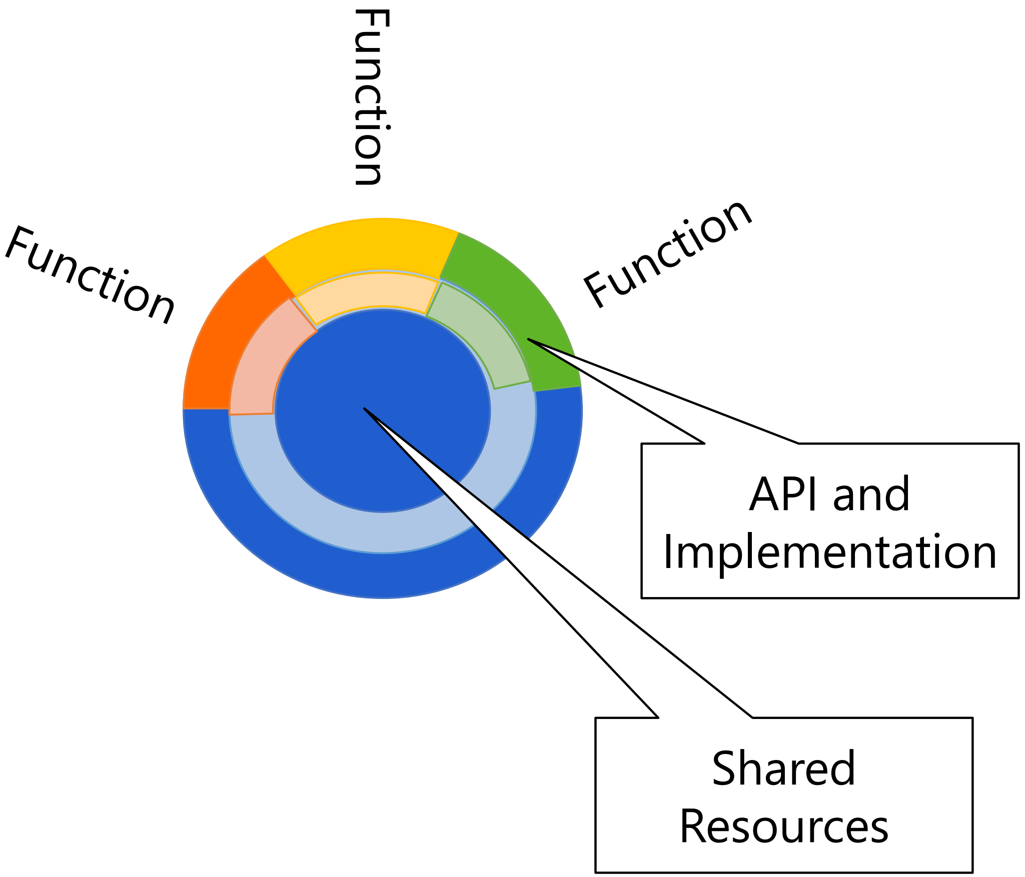
- Report independent, actionable state changes to authorized subscribers
  - “Blob created”
  - “Sales lead created”
  - “Order confirmed”
- Allows simple, noninvasive, reactive extension of the functionality of a service



Enter “Serverless“

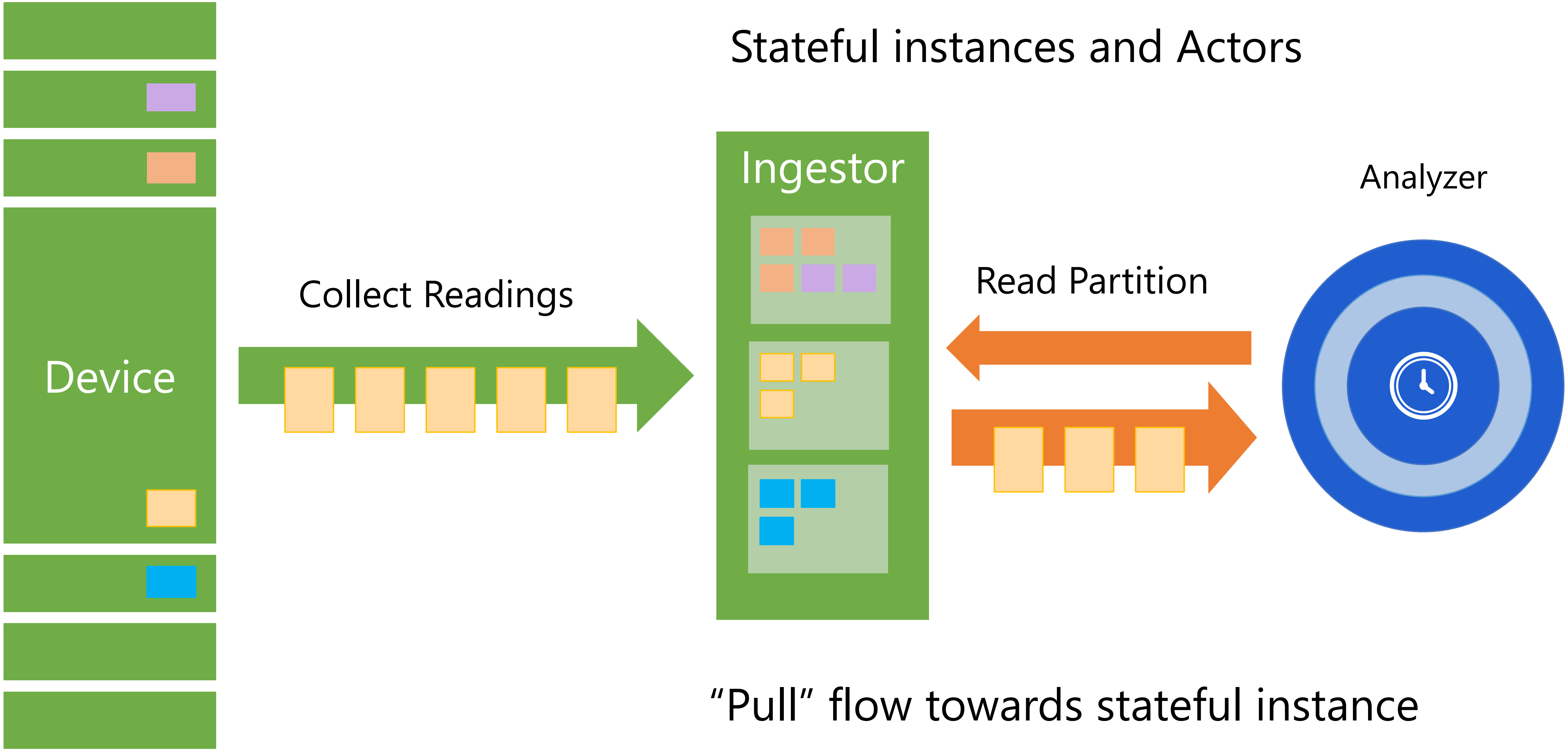
How do Functions/Lambdas fit?

- A service can be made up of a **fleet of** independently deployed **functions** that jointly operate on a shared set of resources
- The service interface is made up from the union of the function interfaces
- The function interfaces may be a mix of RPC-style call interfaces and event driven ones



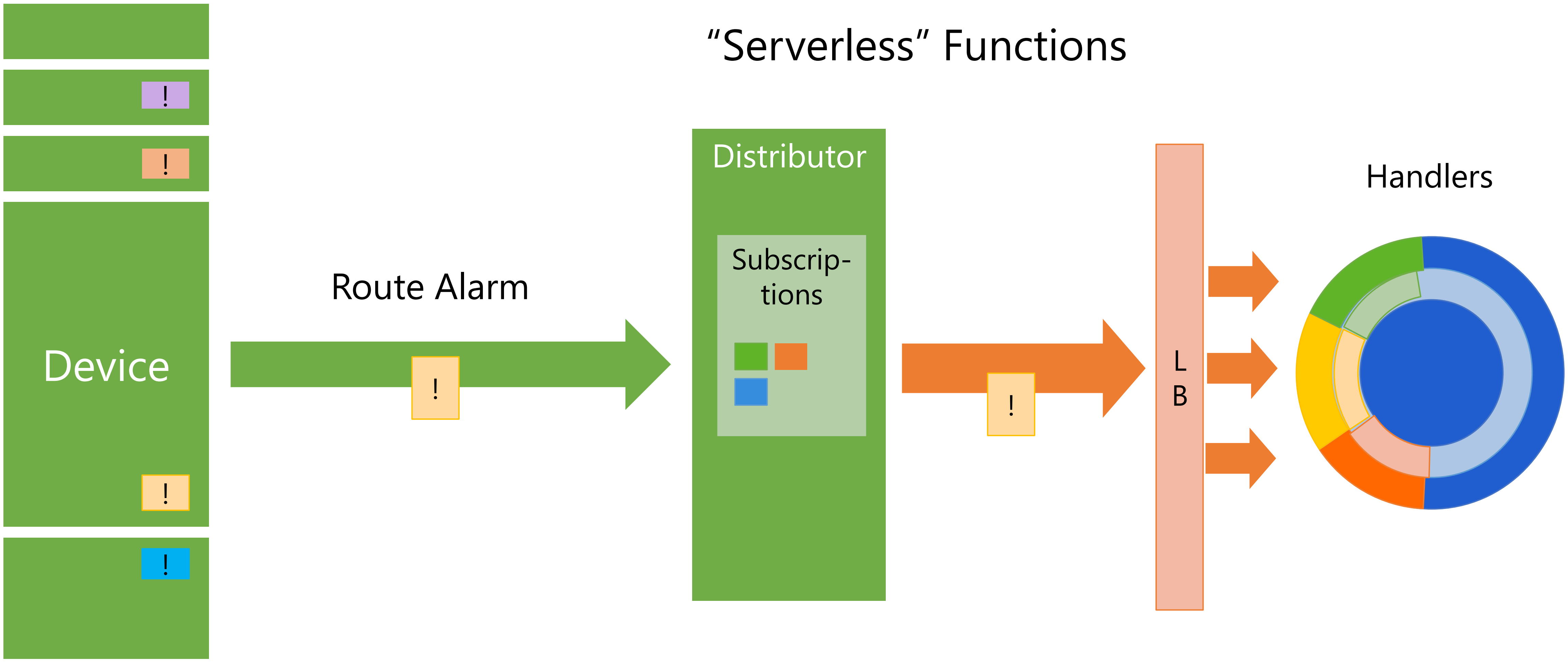
Example: Data Series Processing

Stateful instances and Actors



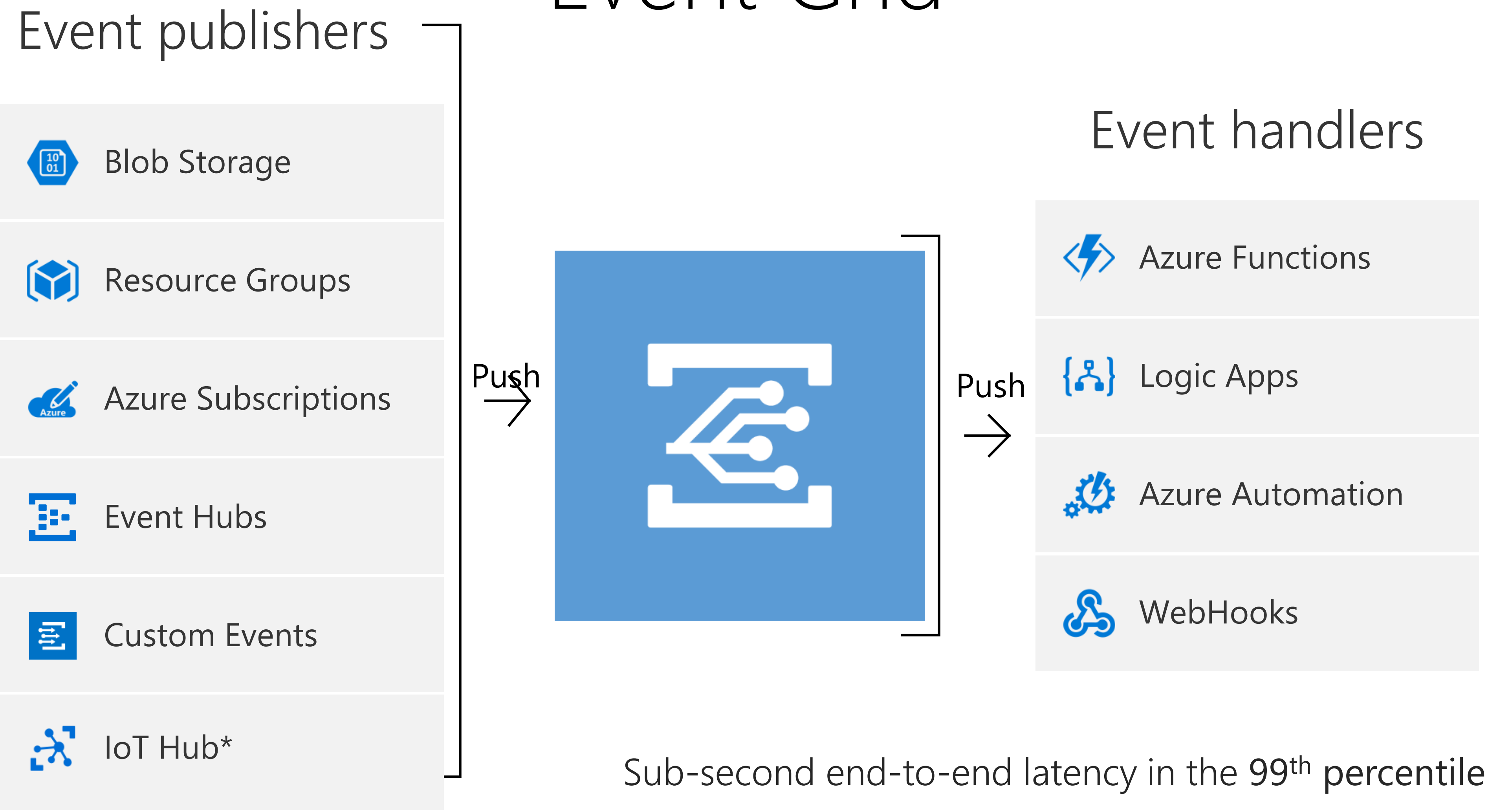
“Pull” flow towards stateful instance

Example: Discrete Event Handling –Alarms



"Push" flow towards stateless handlers

Event Grid



Event publishers

- Blob Storage
- Resource Groups
- Azure Subscriptions
- Event Hubs
- Custom Events
- IoT Hub*

Push

Push

Event handlers

- Azure Functions
- Logic Apps
- Azure Automation
- WebHooks

Sub-second end-to-end latency in the 99th percentile

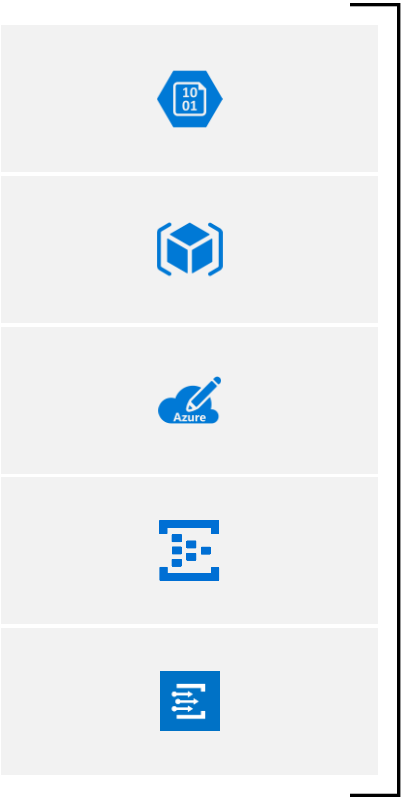
10,000,000 events per second per region

24-hour retry with exponential back off for events not delivered

* Coming soon

Platform-level event plane that’s “just there”

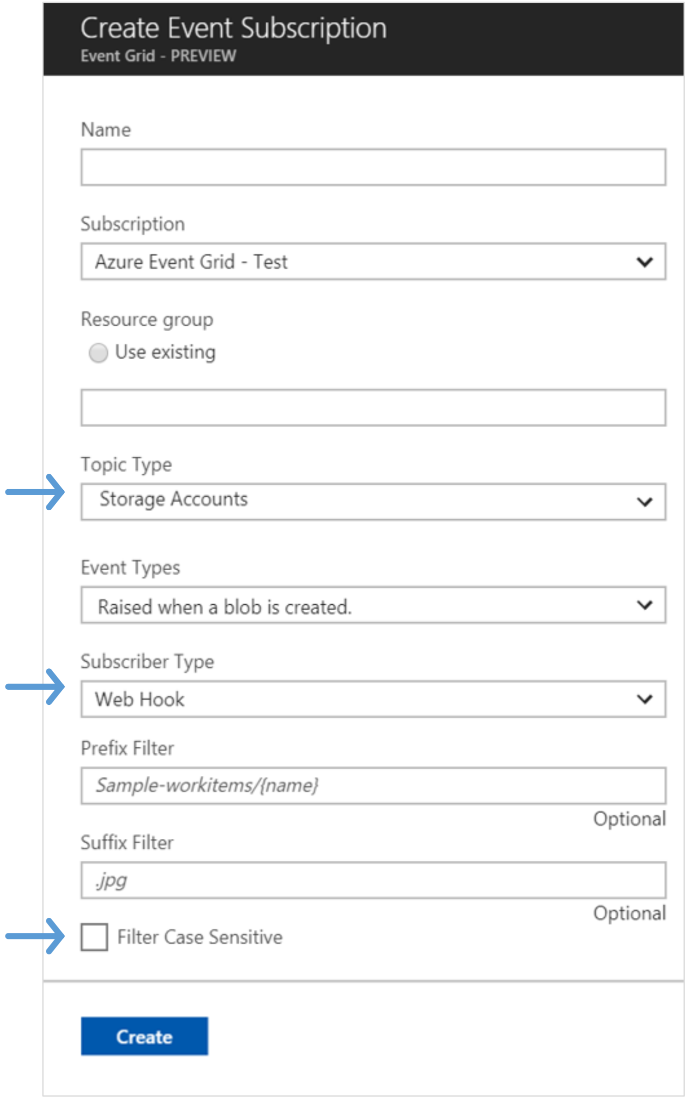
Event publishers



Subscribe to pre-defined system events in Azure or create your own custom topics

Route events to any end-points, Azure or even beyond

Enable filtering and efficient routing of events



Create Event Subscription
Event Grid - PREVIEW

Name

Subscription
Azure Event Grid - Test

Resource group
○ Use existing

Topic Type
Storage Accounts

Event Types
Raised when a blob is created.

Subscriber Type
Web Hook

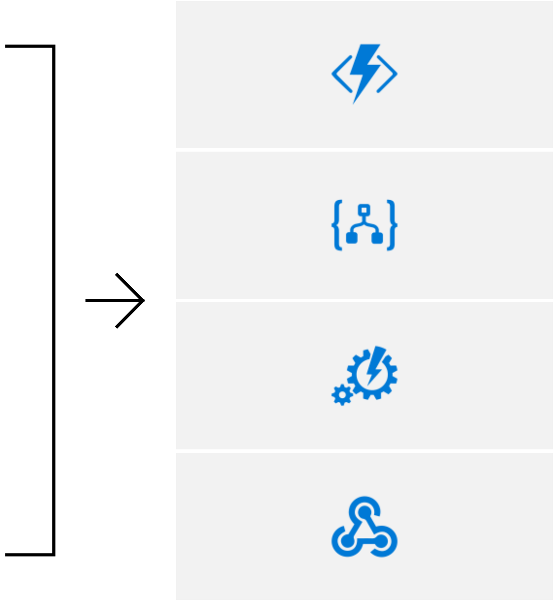
Prefix Filter
Sample-workitems/{name}
Optional

Suffix Filter
.jpg
Optional

□ Filter Case Sensitive

Create

Event handlers



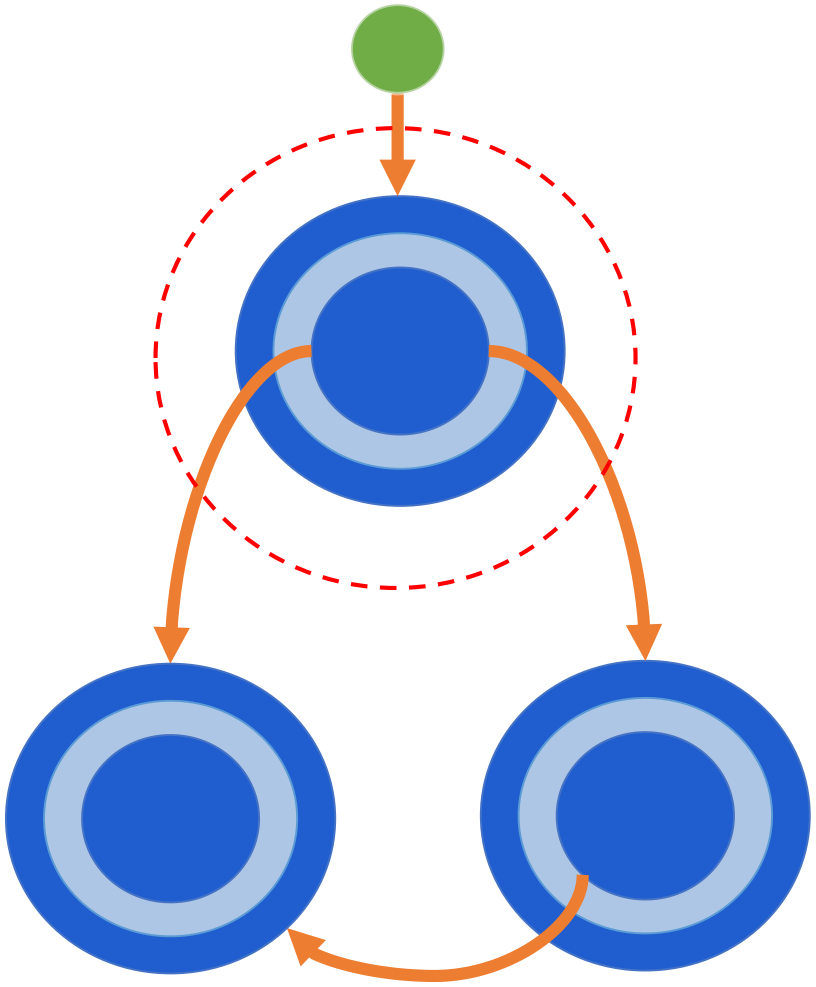
But Lock-In?!

Build twice right.

Building the same solution twice, with shared code, leveraging as much of Azure and AWS PaaS services is operationally cheaper and more reliable than any DIY alternative in most companies’ reach.

A "Service" is software that …

- … is responsible for holding, processing, and/or distributing particular kinds of information within the scope of a system
- … can be built, deployed, and run independently, meeting defined operational objectives
- … communicates with consumers and other services, presenting information using conventions and/or contract assurances
- … protects itself against unwanted access, and its information against loss
- … handles failure conditions such that failures cannot lead to information corruption



goto; copenhagen

GOTO Copenhagen 2017
Conference Oct. 1-3, 2017

Please

Remember to rate this session

Thank you!

Follow us @gotocph

goto; copenhagen

Follow us @gotocph