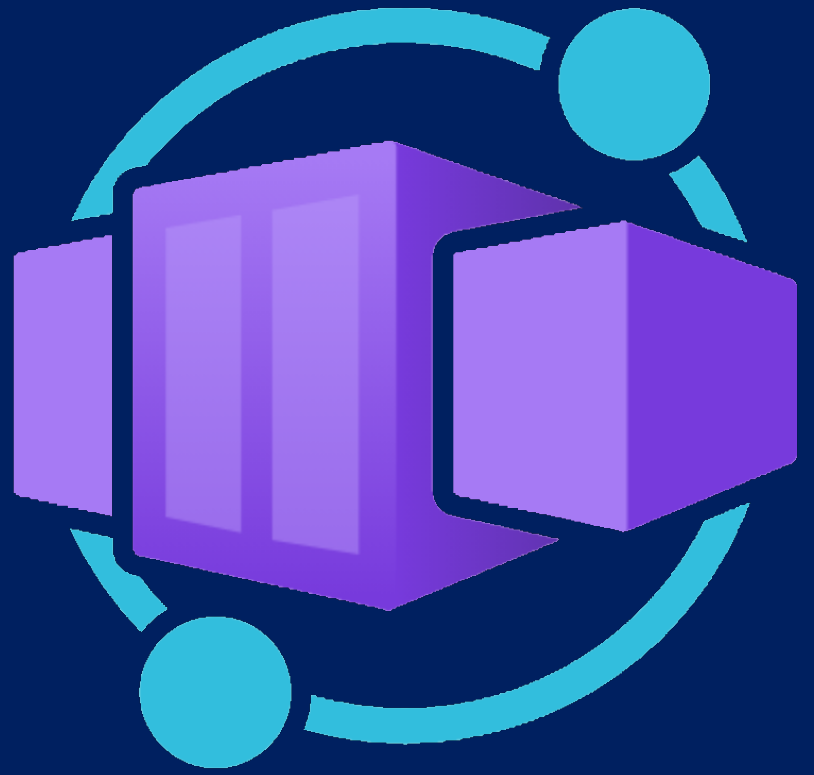
Microsoft

Azure Container Apps

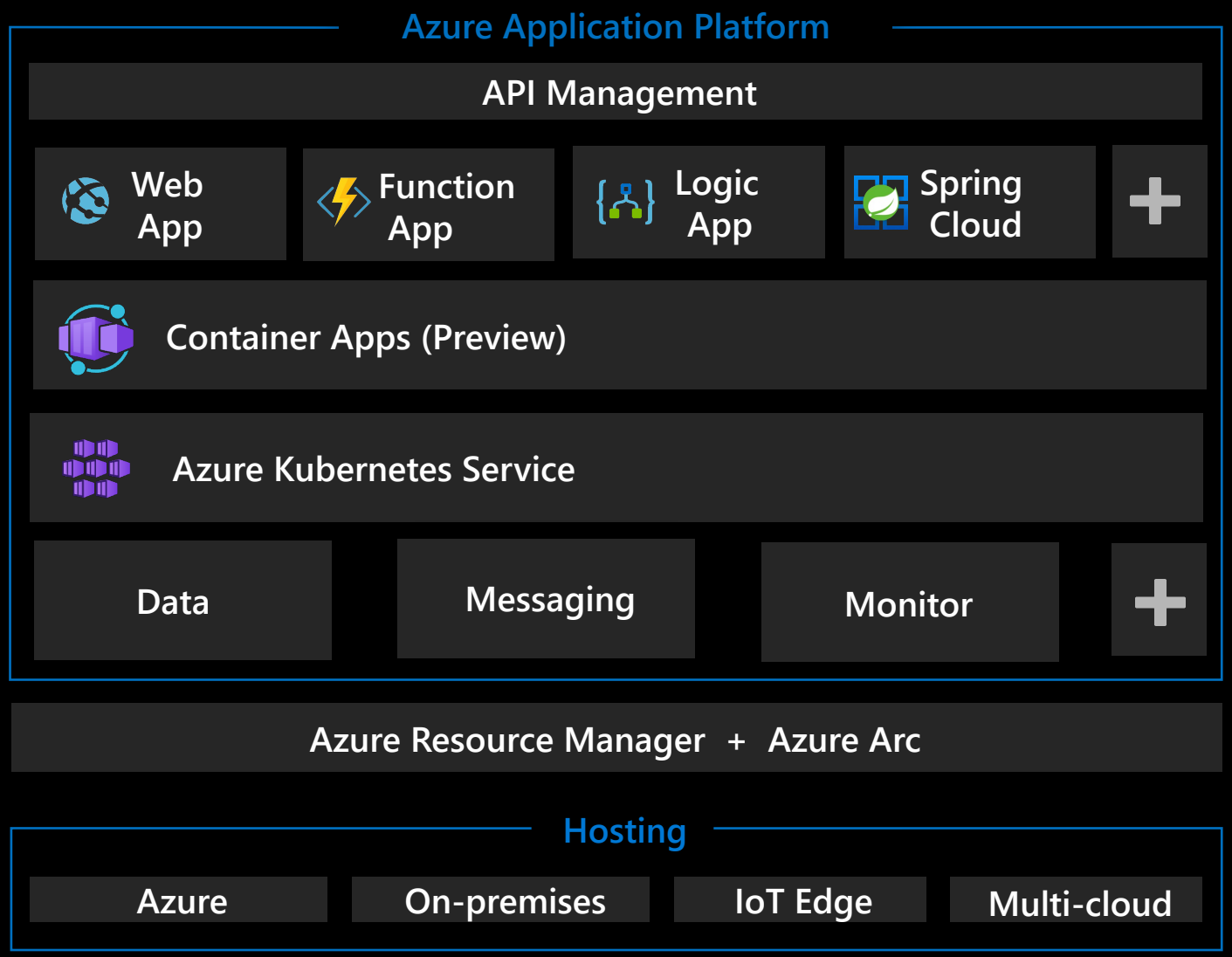
Overview



Arturo Quiroga
Sr. CSA – Azure Core, App Innovation and IoT.
Microsoft Canada – GPS

Joel Hebert
Sr. CSA – Digital & App Innovation.
Microsoft Canada - GPS

Azure's Cloud Native Application Platform



	Azure Container Instances	Azure Kubernetes Service	Azure Red Hat OpenShift	Azure Container Apps	Azure Spring Cloud	Azure App Service	Azure Static Web Apps	Azure Functions
GCP and AWS competing services	AWS Fargate	Google Kubernetes Engine (GKE) AWS Elastic Kubernetes Service (EKS)	Red Hat OpenShift on AWS (ROSA)	Google Cloud Run AWS App Runner		Google App Engine AWS Elastic Beanstalk	Google Firebase Hosting AWS Amplify, S3	Google Functions AWS Lambda
Optimized for	Single container	Containerized applications, APIs, and microservices			Spring Boot applications	Windows and Linux based 3-tier web apps (code, container)	Web frontend apps	Event-driven code
Language affinity	Any language, any framework				Java	.NET, Java, Python, Node.JS, PHP, Ruby	Angular, React, Svelte, Vue, Blazor	.NET, C#, Node.JS, TypeScript, Java, Python, PowerShell
Ecosystem affinity		CNCF	Red Hat		Spring, VMware	Microsoft		Microsoft
Developer experience	• Minimal developer experience after deploying containers. • No support for service discovery, multiple microservices, etc.	• Deploy containers using Kubernetes YAML files • Use Deployment Center to integrate with GitHub Actions. • Use open-source and CNCF projects like Istio, Knative, and ArgoCD, Tekton. • Use select built-in (managed) projects like KEDA, Dapr, and Azure ML.	• Deploy containerized apps using Red Hat Universal Base Images • Bring application code and build using S2I • Use built-in CI/CD with OpenShift Pipelines • Leverage supported Red Hat components like OpenShift Service Mesh and OpenShift Serverless	• Deploy containerized applications or integrate with GitHub Actions. • Use built-in Dapr for microservices communication, pub/sub, etc. • Rely on built-in application lifecycle management features such as revisions and traffic shifting.	• Build Spring Boot applications using Maven and deploy the JAR artifact. • Integrate with GitHub Actions to automate CI/CD workflow.	• Bring your web app code, integrate with Git, or deploy a container image. • Run Kubernetes anywhere across Azure. • Tight integration of Visual Studio and GitHub.	• Build a website with all popular JavaScript frameworks within clicks • Create backend application with Azure Functions or Blazor • Generate staging versions for quick preview before publishing	• Build event-driven functions using the Azure Functions SDK or the Azure Functions base runtime image. • Trigger functions from different event-sources.
Operator experience	• Use Azure APIs/portal to deploy a single container. • No support for scaling.	• Use Kubernetes native APIs/Azure portal to create and manage workloads. • Integrate with Azure services like Container Registry. • Configure scaling using cluster autoscaler, burst using Virtual Nodes to ACI. • Apply governance policies using Azure Policy.	• Use OpenShift native APIs/OpenShift console to create and manage workloads. • Use the Operator Hub to enable additional functionality such as OpenShift Serverless. • Configure scaling using Machine Set scaler.	• Use Azure APIs/portal to create a container app environment which hosts multiple apps. • Define scaling parameters using KEDA with scale-to-zero support.	• Integrated logging and monitoring with Azure Log Analytics and Application Insights (or 3rd party APMs). • Autoscaling based on schedule and application metrics. • Support for green/blue deployment strategy.	• Azure Web Apps can be deployed on Kubernetes if required using Azure Arc. • Simple operation model and fully managed experience. • Reduce downtime and minimize risk by using deployment slots	• Custom Domain management and free SSL Certificates autorenewal for best security coverage. • Authentication integration with GitHub, AAD and Twitter accounts.	• [..]. Azure Functions can be deployed on Kubernetes if required using Azure Arc.

How does ACA compare to AKS?

	Azure Kubernetes Service (AKS) Infrastructure focus, higher flexibility	Azure Container Apps (ACA) Application focus, infrastructure abstraction
Core value proposition	Managed Kubernetes cluster in Azure with full access to the Kubernetes API server and high level of control over cluster configuration with a node-based pricing model	Fully-managed serverless abstraction on top of Kubernetes infrastructure, purpose built for managing and scaling event-driven microservices with a consumption-based pricing model
Optimized for	• Upstream feature parity with a managed control plane • Operations flexibility with advanced customization • Experienced Kubernetes operators	• Platform-as-a-Service experience with serverless scale • Developer productivity with low operations overhead • Linux-based, general-purpose stateless containers
Interaction model	• Operators deploy node-based AKS clusters using Azure Portal, CLI or Infrastructure-as-Code templates (IaC) • Developers deploy containers via Kubernetes deployment manifests or HELM charts to logically-isolated namespaces within the cluster	• Developers deploy containers as individual Container Apps using Azure Portal, CLI or IaC templates without any Kubernetes manifests required • Related container apps are deployed to a shared Container Apps environment comparable to a Kubernetes namespace
OSS Integration	• Provides a set of cluster extensions and add-ons for operators to enable OSS components in-cluster including Dapr, KEDA, Open Service Mesh, GitOps (Flux), Pod Identity, etc. • Supports manual installation via Kubernetes manifests	Includes opinionated platform capabilities powered by CNCF projects including Dapr, KEDA and Envoy which are fully platform-managed and supported • Envoy: managed ingress and traffic splitting • KEDA: managed, event-driven autoscale • Dapr: codified best practices for microservices

Azure Container Apps

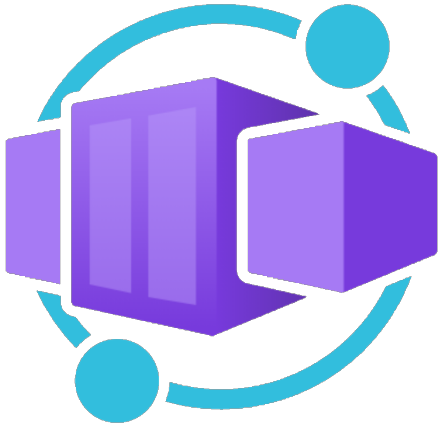
Azure Container Apps enable you to run microservices and containerized applications on a serverless platform.

Common uses of Azure Container Apps include:

- Deploying **API endpoints**
- Hosting **background processing applications**
- Handling **event-driven processing**
- Running **microservices**

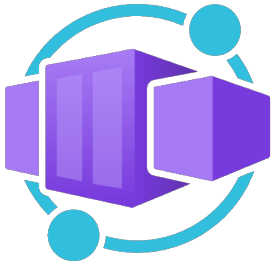
Applications built on Azure Container Apps can **dynamically scale** based on the following characteristics:

- HTTP traffic
- Event-driven processing
- CPU or memory load
- Any KEDA-supported scaler



Built on a foundation of AKS, KEDA, Dapr, and Envoy

Azure Container Apps (ACA)



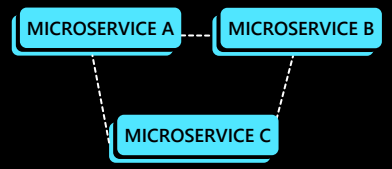
With Azure Container Apps, you can:

- Run multiple container **revisions** and manage the container app's application lifecycle.
- **Autoscale** your apps based on any KEDA-supported scale trigger. Most applications can scale to zero.
- Enable **HTTPS ingress** without having to manage other Azure infrastructure.
- **Split traffic** across multiple versions of an application for Blue/Green deployments and A/B testing scenarios.
- Use **internal ingress** and **service discovery** for secure internal-only endpoints with built-in DNS-based service discovery.
- Build microservices with **Dapr** and access its rich set of APIs.
- Run containers from any registry, public or private, including Docker Hub and Azure Container Registry (ACR).
- Use the Azure CLI extension or ARM templates to manage your applications.
- Provide an existing **virtual network** when creating an environment for your container apps.
- Securely **manage secrets** directly in your application.
- View **application logs** using Azure Log Analytics.
- **Generous quotas** which can be overridden to increase limits on a per-account basis

NOTE: Applications that scale on CPU or memory load can't scale to zero. Applications that scale based on event-driven or HTTP requests will scale to 0.

What can you build with Azure Container Apps?

Microservices



Deploy and manage a microservices architecture with the option to integrate with Dapr.

AUTO-SCALE CRITERIA

Individual microservices can scale independently using any KEDA scale triggers

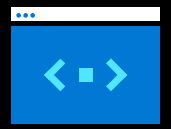
Event-driven processing

E.g., queue reader application that processes messages as they arrive in a queue.

AUTO-SCALE CRITERIA

Scaling is determined by the number of messages in the queue

Web Applications

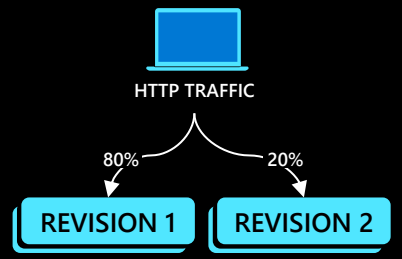


Deploy web apps with custom domains, TLS certificates, and integrated authentication.

AUTO-SCALE CRITERIA

Scaling is determined by the number of concurrent HTTP requests

Public API endpoints



HTTP requests are split between two revisions of the app — the first revision gets 80% of the traffic, while a new revision receives 20%.

AUTO-SCALE CRITERIA

Scaling is determined by the number of concurrent HTTP requests

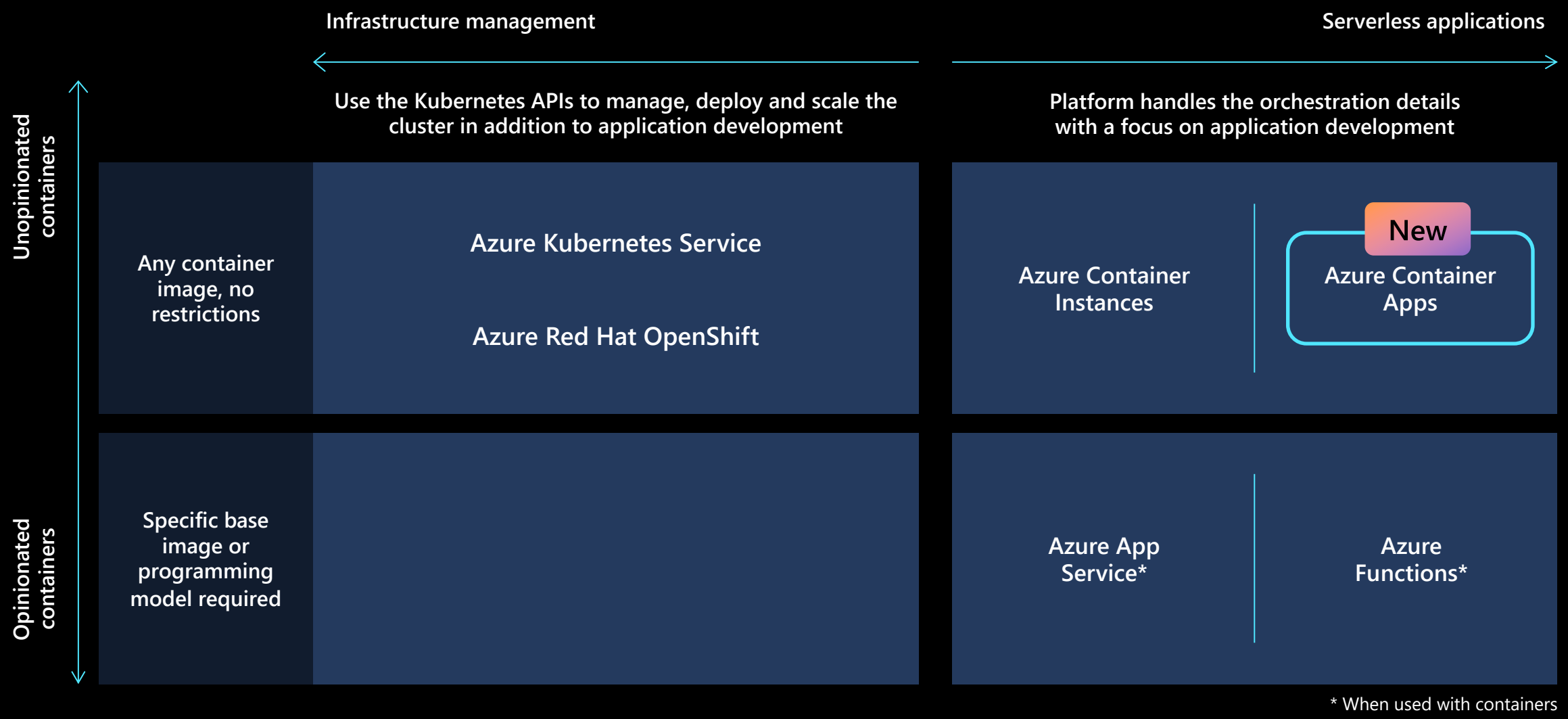
Background processing

E.g., continuously-running background process that transforms data in a database.

AUTO-SCALE CRITERIA

Scaling is determined by the level of CPU or memory load

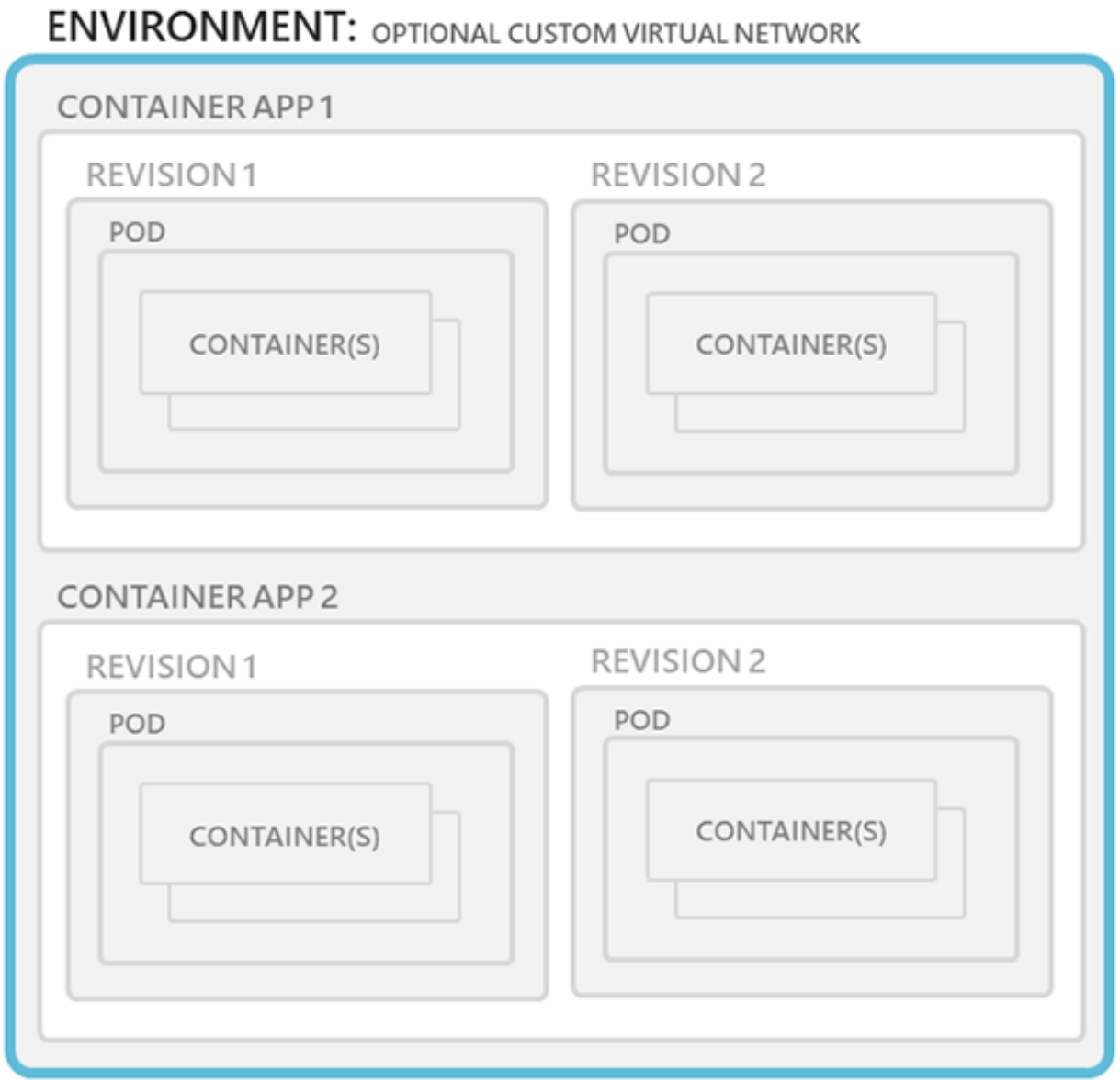
Azure containers portfolio



* When used with containers

Azure Container Apps Environments

Environments are an isolation boundary around a collection of container apps.



- Individual container apps are deployed to a single Container Apps environment, which acts as a secure boundary around groups of container apps.
- Container Apps in the same environment are deployed in the same virtual network and write logs to the same Log Analytics workspace.

How to deploy Container Apps?

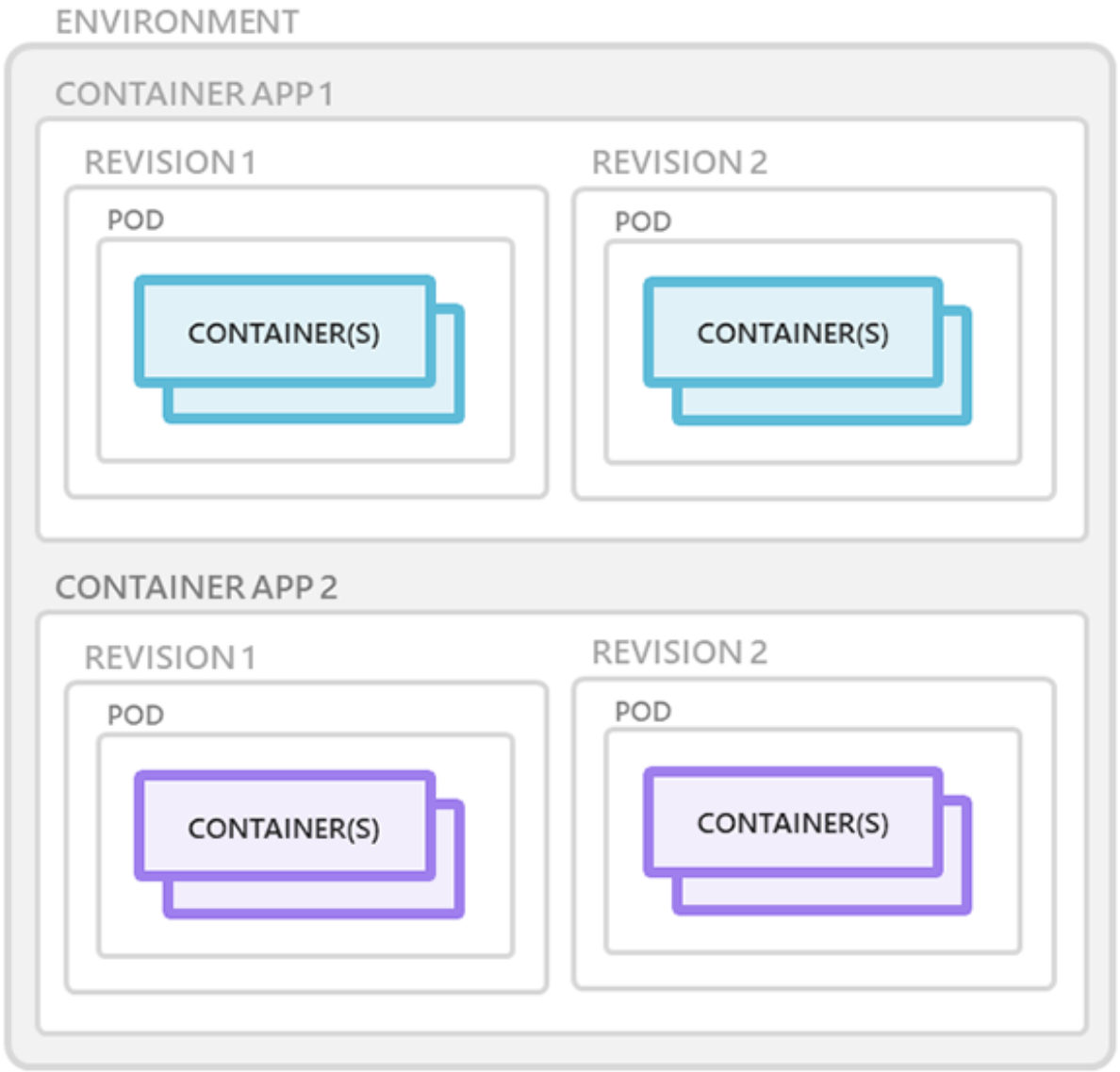
- Reasons to deploy container apps to the **same environment** include situations when you need to:
  - Manage related services
  - Deploy different applications to the same virtual network
  - Have applications communicate with each other using Dapr
  - Have applications to share the same Dapr configuration
  - Have applications share the same log analytics workspace

- Reasons to deploy container apps to **different environments** include situations when you want to ensure:
  - Two applications never share the same compute resources
  - Two applications can't communicate with each other via Dapr

Containers in Azure Container Apps.

- Azure Container Apps manages the details of Kubernetes and container orchestration for you.
- Containers in Azure Container Apps can use any runtime, programming language, or development stack of your choice.

Containers for an Azure Container App are grouped together in pods inside revision snapshots.



Azure Container Apps Supports:

- Azure Container Apps supports:
  - Any **Linux-based** x86-64 (linux/amd64) container image
  - Containers from any public or private container registry
- Features include:
  - There's no required base container image.
  - Changes to the template ARM configuration section trigger a new container app revision.
  - If a container crashes, it automatically restarts.
- **Note**
  - The only supported protocols for a container app's fully qualified domain name (FQDN) are **HTTP and HTTPS** through ports 80 and 443 respectively.

Multiple Containers

You can define multiple containers in a single container app.

The containers in a container app share hard disk and network resources and experience the same application lifecycle.

To run multiple containers in a container app, add more than one container in the containers array of the container app template.

Reasons to run containers together in a container app include:

- Use a container as a sidecar to your primary app.
- Share disk space and the same virtual network.
- Share scale rules among containers.
- Group multiple containers that need to always run together.
- Enable direct communication among containers.

ACA Limitations

Azure Container Apps has the following limitations:

- **Privileged containers**: Azure Container Apps can't run privileged containers. If your program attempts to run a process that requires root access, the application inside the container experiences a runtime error.

- **Operating system**: Linux-based (linux/amd64) container images are required.

ACA Quotas

The following quotas are on a per subscription basis for Azure Container Apps.

- To request an increase in quota amounts for your container app, you need to submit a support ticket.

Feature	Quantity
Environments per region	15
Container apps per environment	Unlimited
Revisions per container app	100
Replicas per container app	30
Cores per replica	2
Cores per environment	40

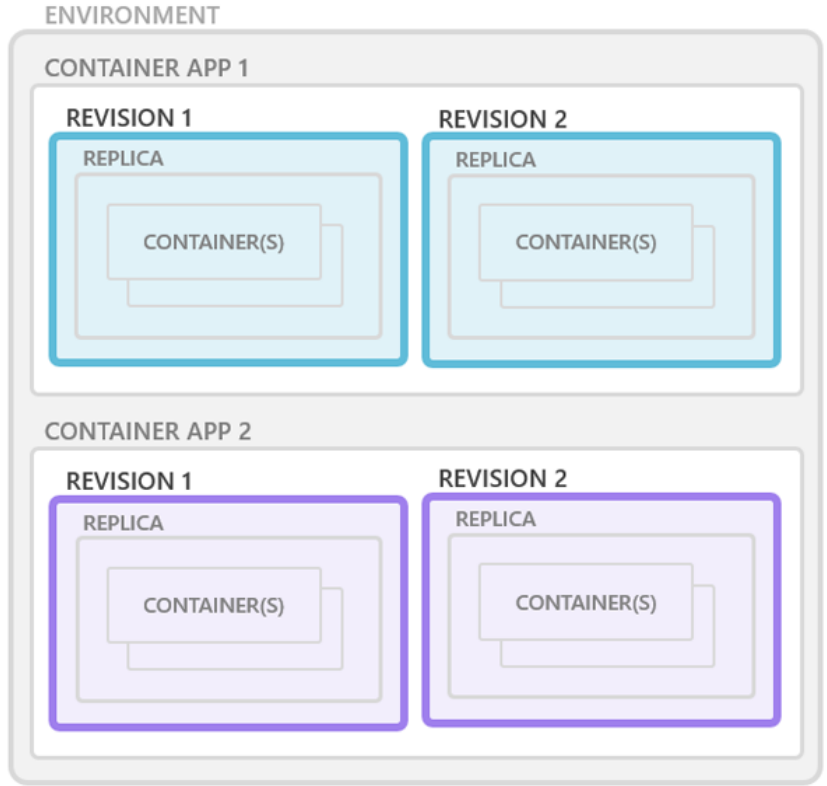
Quotas for Azure Container Apps | Microsoft Learn

Revisions in Azure Container Apps (1)

Azure Container Apps implements container app versioning by creating revisions.

- The first revision is automatically created when you deploy your container app.
- New revisions are automatically created when you make a *revision-scope* change to your container app.
- While revisions are immutable, they're affected by *application-scope* changes, which apply to all revisions.
- You can retain up to 100 revisions, giving you a historical record of your container app updates.
- You can run multiple revisions concurrently.
- You can split external HTTP traffic between active revisions.

Revisions are immutable snapshots of a container app.

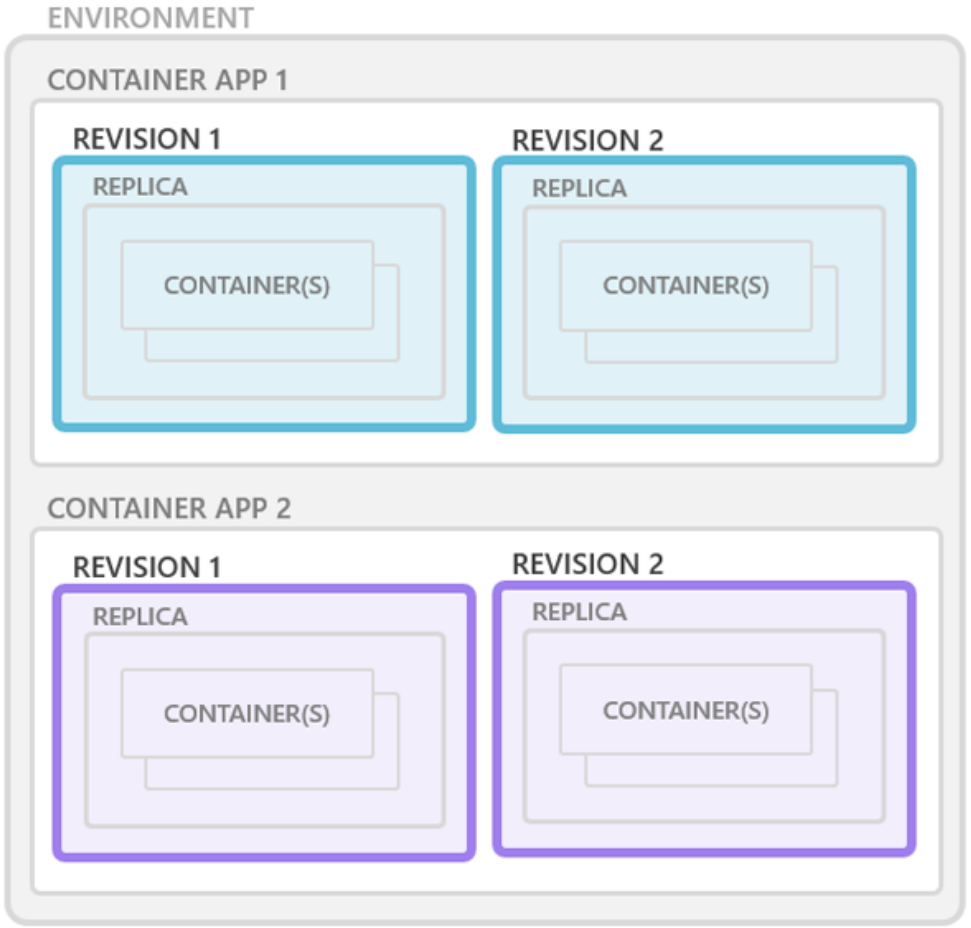


Revisions in Azure Container Apps (2)

Use cases

- Container Apps revisions help you manage the release of updates to your container app by creating a new revision each time you make a *revision-scope* change to your app.
- You can control which revisions are active, and the external traffic that is routed to each active revision.
- You can use revisions to:
  - Release a new version of your app.
  - Quickly revert to an earlier version of your app.
  - Split traffic between revisions for A/B testing.
  - Gradually phase in a new revision in blue-green deployments. For more information about blue-green deployment, see BlueGreenDeployment.

Revisions are immutable snapshots of a container app.



Revisions in Azure Container Apps (3)

Application Lifecycle:

Deployment:

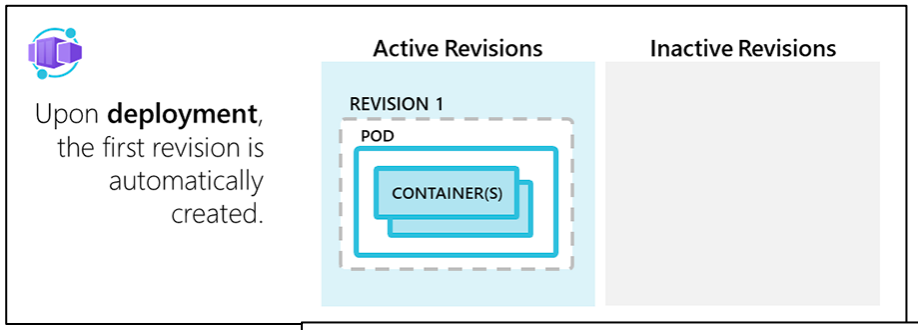
- As a container app is deployed, the first revision is automatically created.

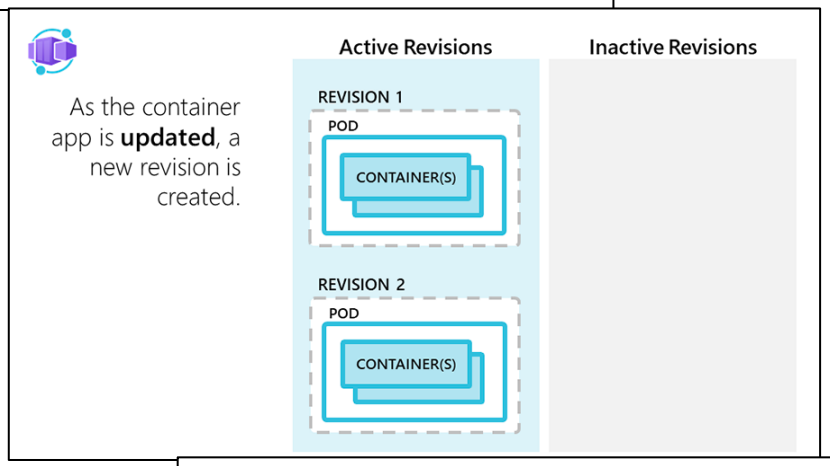
Update:

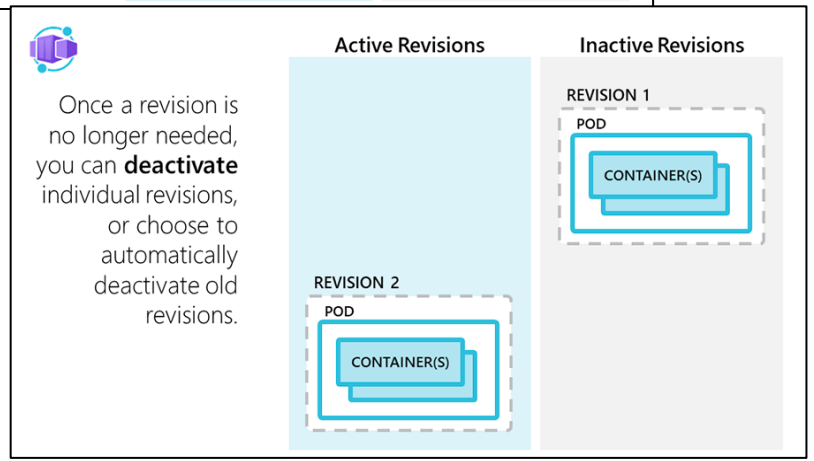
- As a container app is updated with a revision scope-change, a new revision is created.
- You can choose whether to automatically deactivate new old revisions, or allow them to remain available.

Deactivate:

- Once a revision is no longer needed, you can deactivate a revision with the option to reactivate later.
- During deactivation, containers in the revision are shut down.

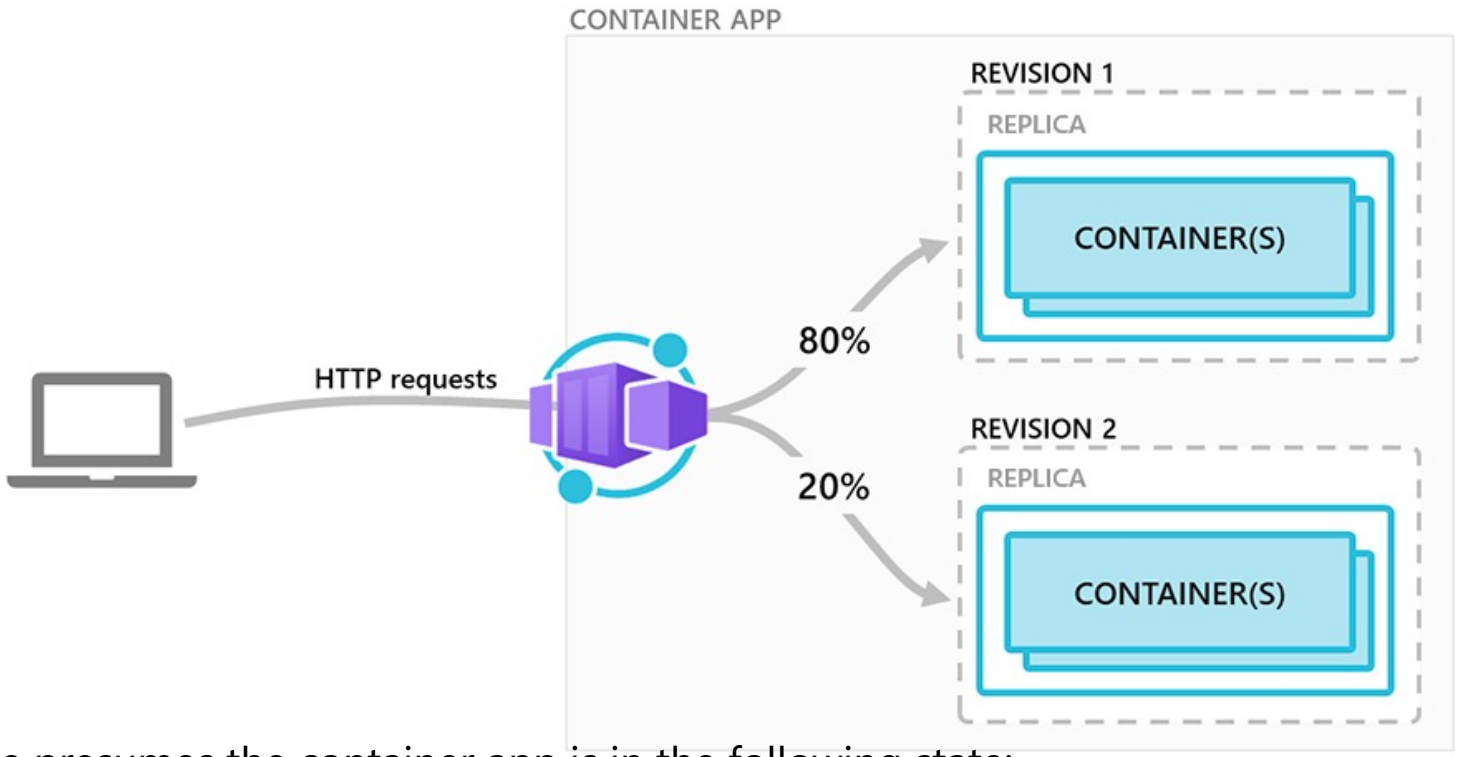






Revisions in Azure Container Apps (4)

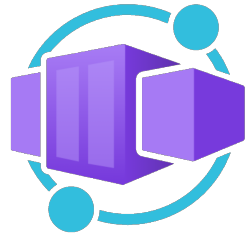
The following diagram shows a container app with two revisions



The scenario shown above presumes the container app is in the following state:

- Ingress is enabled, making the container app available via HTTP.
- The first revision was deployed as *Revision 1*.
- After the container was updated, a new revision was activated as *Revision 2*.
- Traffic splitting rules are configured so that *Revision 1* receives 80% of the requests, and *Revision 2* receives the remaining 20%.

Revisions in Azure Container Apps (5)



Change types

- Changes to a container app fall under two categories: *revision-scope* or *application-scope* changes.

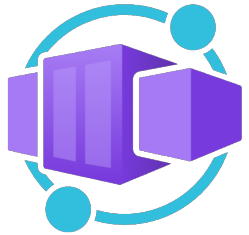
- *Revision-scope* changes trigger a new revision when you deploy your app, while *application-scope* changes don't.
- **Revision-scope changes**

- A new revision is created when a container app is updated with *revision-scope* changes. The changes are limited to the revision in which they're deployed, and don't affect other revisions.

- A ***revision-scope*** change is any change to the parameters in the properties.template section of the container app resource template.

These parameters include:
- Revision suffix
- Container configuration and images
- Scale rules for the container application

Revisions in Azure Container Apps (6)



Application-scope changes

When you deploy a container app with *application-scope* changes:

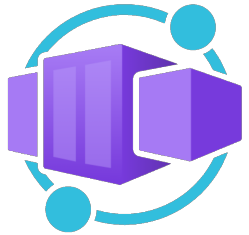
- The changes are globally applied to all revisions.
- A new revision isn't created.

Application-scope changes are defined as any change to the parameters in the properties.configuration section of the container app resource template.

These parameters include:

- Secret values (revisions must be restarted before a container recognizes new secret values)
- Revision mode
- Ingress configuration including:
  - Turning ingress on or off
  - Traffic splitting rules
  - Labels
- Credentials for private container registries
- Dapr settings

Revisions in Azure Container Apps (7)



Revision modes

The revision mode controls whether only a single revision or multiple revisions of your container app can be simultaneously active.

- You can set your app's revision mode from your container app's **Revision management** page in the Azure portal, using Azure CLI commands, or in the ARM template.

Single revision mode

- By default, a container app is in ***single revision mode***. In this mode, only one revision is active at a time. When a new revision is created, the latest revision replaces the active revision.

Multiple revision mode

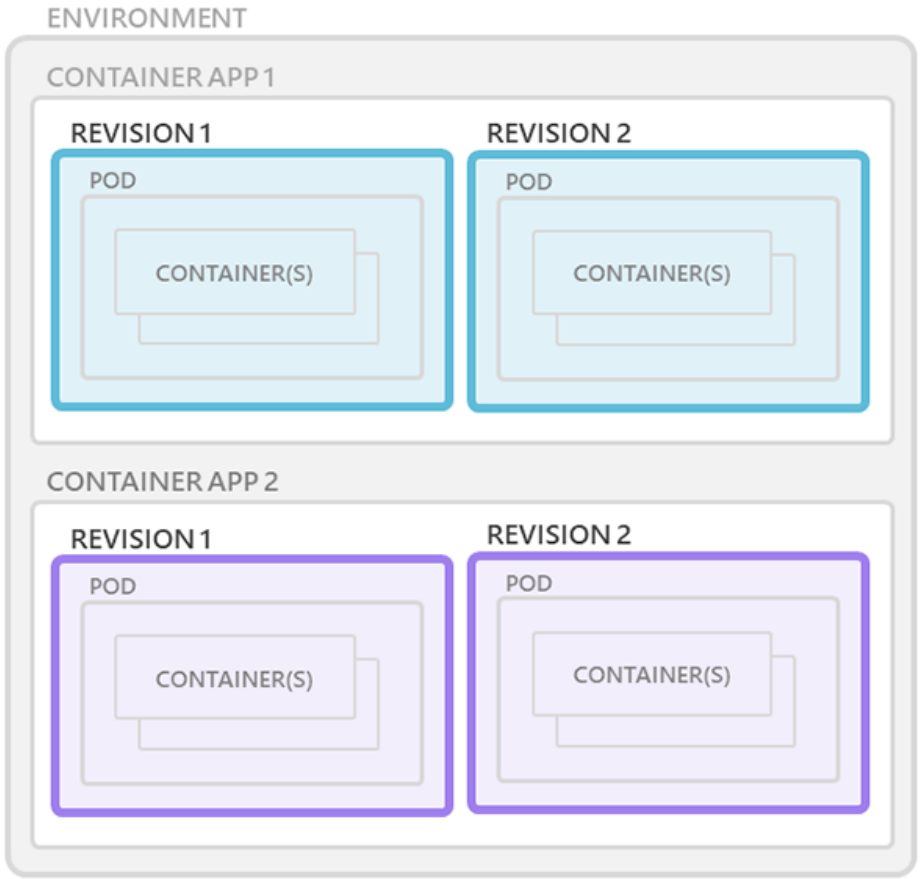
- Set the revision mode to ***multiple revision mode***, to run multiple revisions of your app simultaneously. While in this mode, new revisions are activated alongside current active revisions.

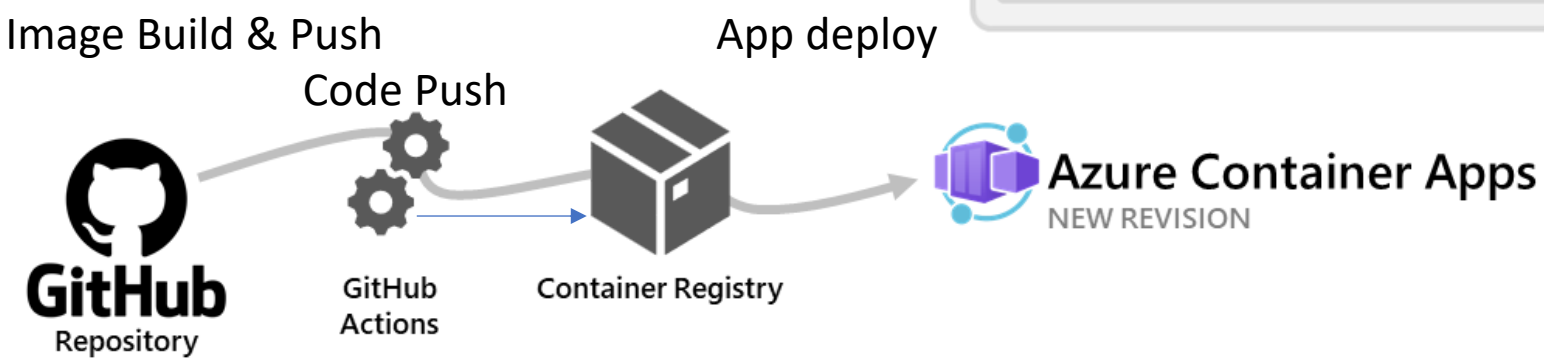
- For an app implementing external HTTP ingress, you can control the percentage of traffic going to each active revision from your container app's **Revision management** page in the Azure portal, using Azure CLI commands, or in an ARM template. For more information, see Traffic splitting.

Publish Revisions with GitHub Actions.

- Azure Container Apps allows you to use GitHub Actions to publish revisions to your container app.
- As commits are pushed to your GitHub repository, a GitHub Action is triggered which updates the container image in the container registry.

Revisions are immutable snapshots of a container app.



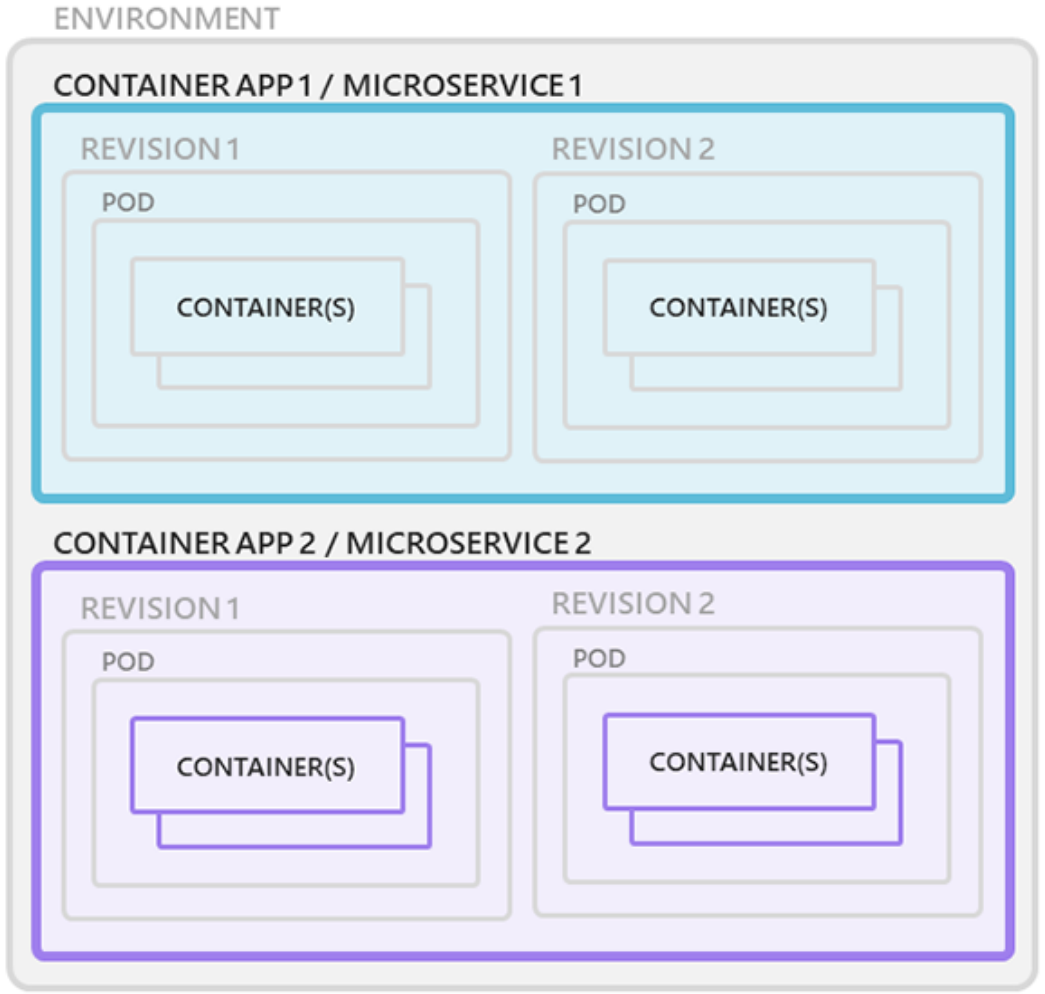


Microservices with Azure Container Apps

Azure Container Apps provides the foundation for deploying microservices featuring:

- Independent scaling, versioning, and upgrades
- Service discovery
- Native Dapr integration

Container apps are deployed as **microservices**.

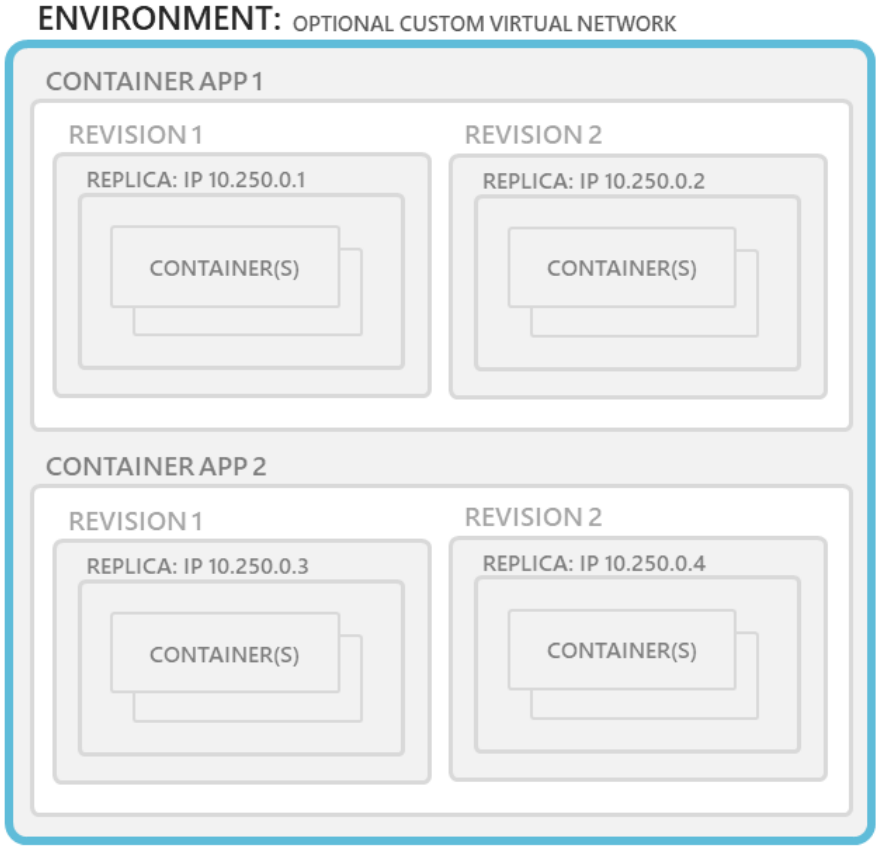


- A Container Apps environment provides a security boundary around a group of container apps.
- A single container app typically represents a microservice, which is composed of container apps made up of one or more containers.

Azure Container Apps – Networking (1)

- Azure Container Apps run in the context of an environment, which is supported by a virtual network (VNET).
- When you create an environment, you can provide a custom VNET, otherwise a VNET is automatically generated for you.
- Generated VNETs are inaccessible to you as they're created in Microsoft's tenent.
- To take full control over your VNET, provide an existing VNET to Container Apps as you create your environment.

A **virtual network** creates a secure boundary around your Azure Container Apps environment.



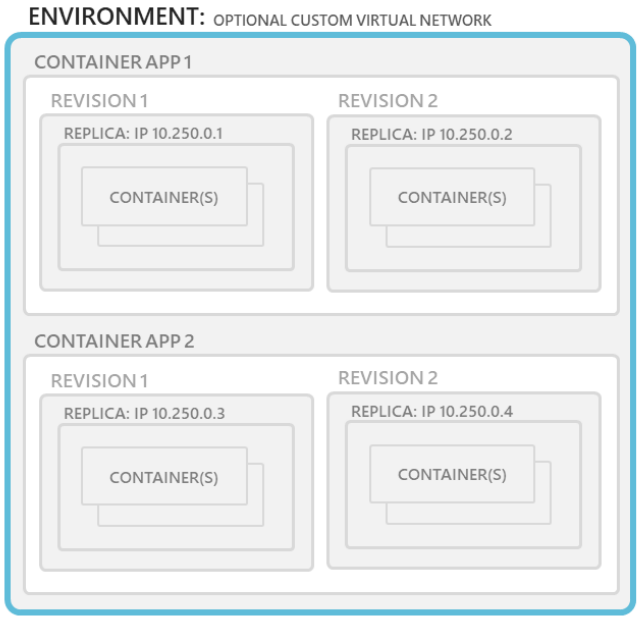
Azure Container Apps – Networking (2)

Custom VNET configuration

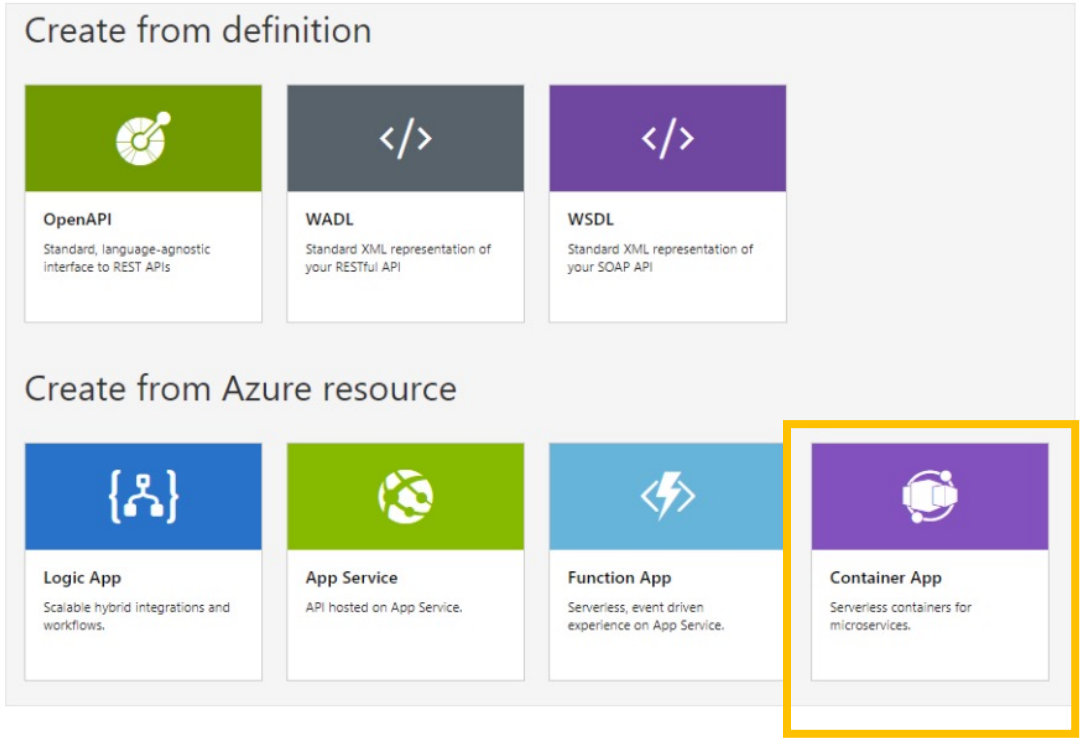
As you create a custom VNET, keep in mind the following situations:

- If you want your container app to restrict all outside access, create an internal Container Apps environment.
- When you provide your own VNET, the network needs a single subnet.
- Network addresses are assigned from a subnet range you define as the environment is created.
  - You can define the subnet range used by the Container Apps environment.
  - Once the environment is created, the subnet range is immutable.
  - A single load balancer and single Kubernetes service are associated with each container apps environment.
  - Each revision is assigned an IP address in the subnet.
  - You can restrict inbound requests to the environment exclusively to the VNET by deploying the environment as internal.

A **virtual network** creates a secure boundary around your Azure Container Apps environment.



API Management Import

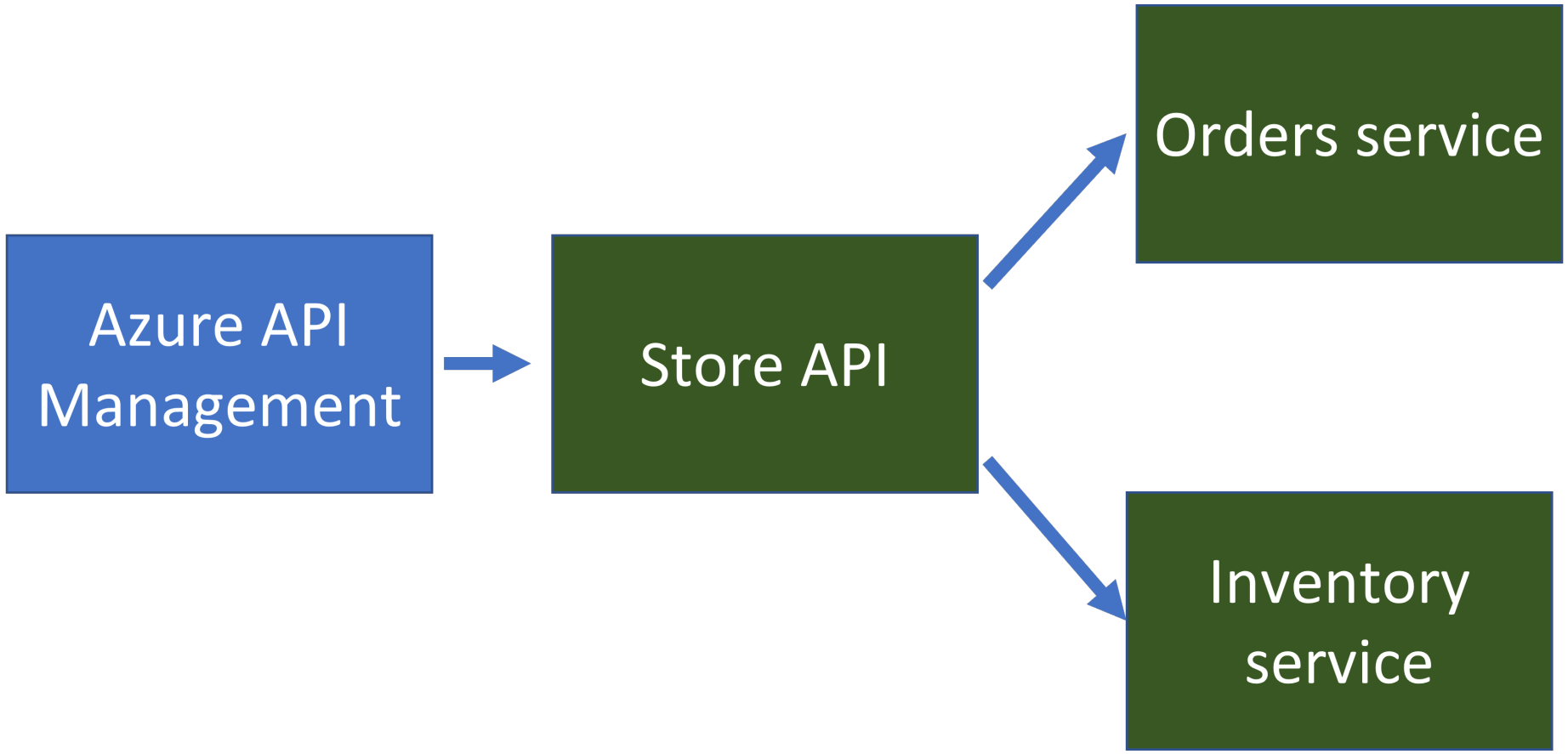


API Management will look in several locations for an OpenAPI Specification:

- The Container App configuration
- /openapi.json
- /openapi.yml
- /swagger/v1/swagger.json

https://docs.microsoft.com/en-us/azure/api-management/import-container-app-with-oas

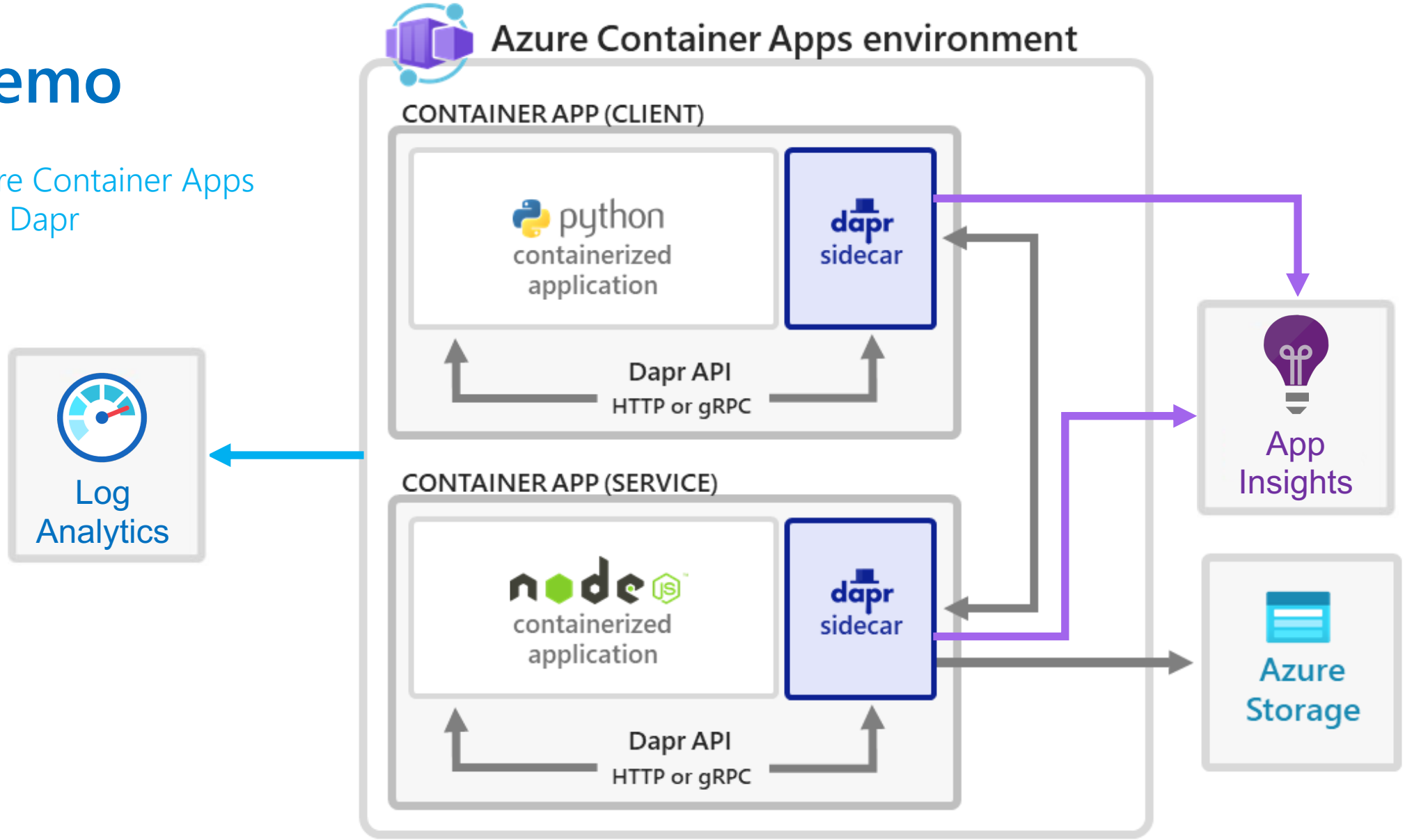
Container Apps Sample



[Container App Store Microservice Sample]

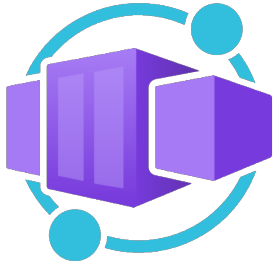
Demo

Azure Container Apps with Dapr



https://github.com/clarenceb/tutorial-dapr-cli

Learn more about Container Apps



- Introducing Azure Container Apps: a serverless container service for running modern apps at scale (Microsoft Tech Community)
- Azure Container Apps Preview documentation
- Azure Container Apps product page
- Container App Store Microservice Sample (GitHub)

KEDA primer

Application autoscaling made simple

Open-source, extensible, and vendor agnostic

KEDA

Kubernetes-based Event Driven Autoscaler

Drive the scaling of any container based on a growing list of 35+ event sources, known as: scalers

Metrics Adapter | Controller | Scaler

Event-driven

Intelligently scale your event-driven applications

Built-in scalers

Out-of-the-box scalers for various vendors, databases, messaging systems, telemetry systems, CI/CD, and more

Vendor-agnostic

Support for triggers across variety of cloud providers & products

Rich capabilities

Bring rich scaling to every workload

keda.sh

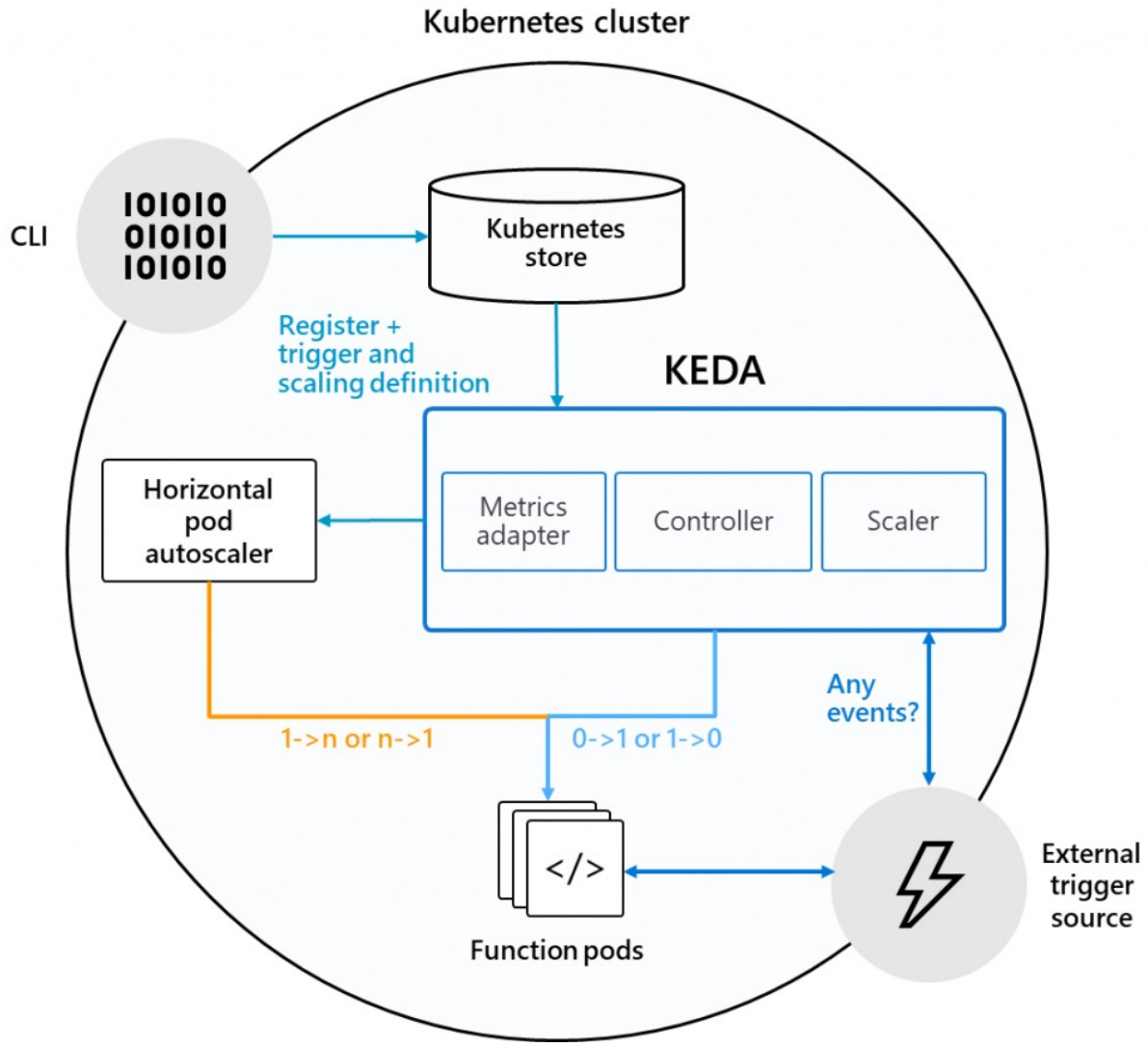
CLOUD NATIVE COMPUTING FOUNDATION

KEDA - Kubernetes Event-driven Autoscaling

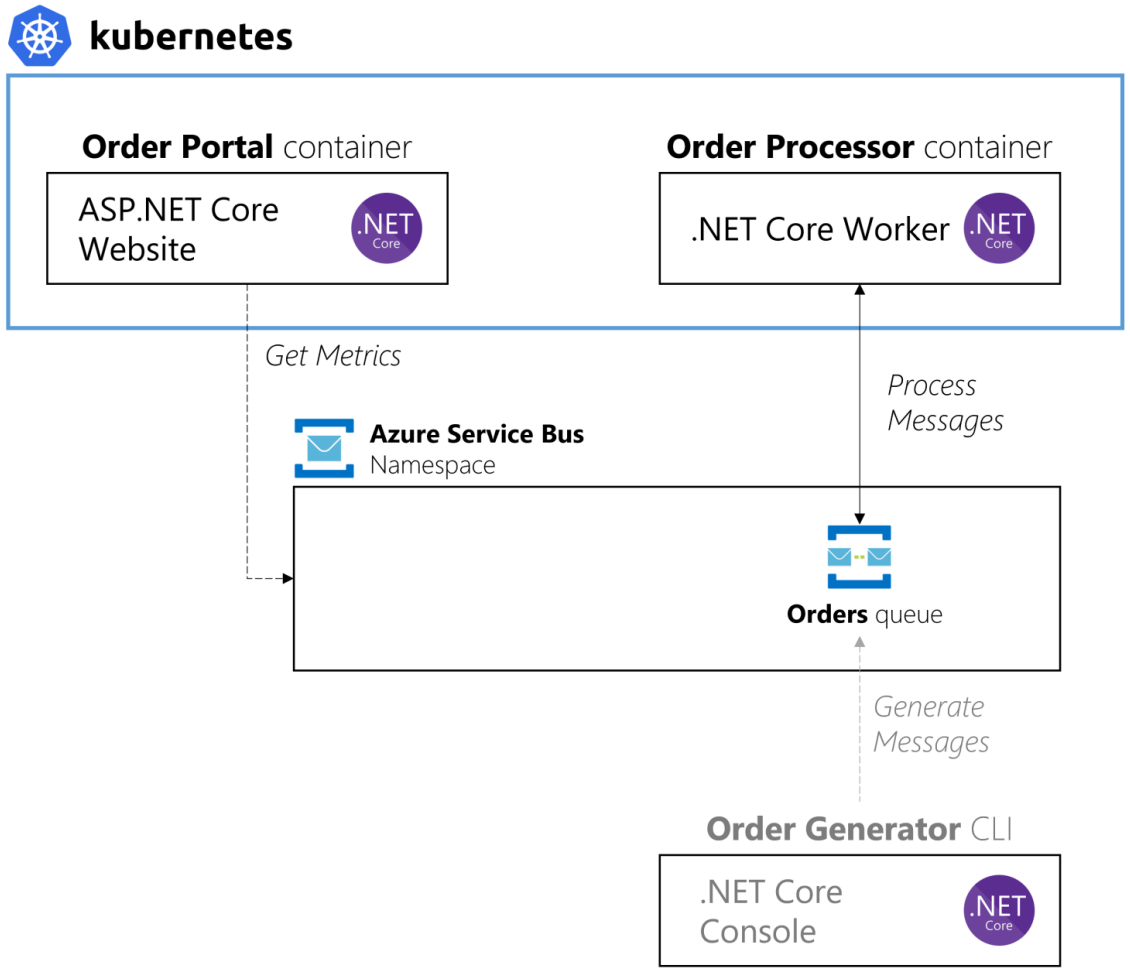
https://keda.sh/
https://www.cncf.io/projects/keda/

- Supports building event-driven applications in Kubernetes
- Fine grained autoscaling off of event sources for *any* container in Kubernetes
- Runs *anywhere* Kubernetes/OpenShift runs
- Native integration with Horizontal Pod Autoscaler (HPA)
- Supports scaling via Jobs (1 event -> 1 job)
- Pods get direct access to event sources
- New hosting option for Azure Functions via containers in Kubernetes
- Built in conjunction with Red Hat
- CNCF incubating project

Architecture



Example




```yaml
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  name: order-processor-scaler
  labels:
    app: order-processor
    name: order-processor
spec:
  scaleTargetRef:
    name: order-processor
  # minReplicaCount: 0 Change to define how many minimum replicas you want
  maxReplicaCount: 10
  triggers:
  - type: azure-servicebus
    metadata:
      queueName: orders
      queueLength: '5'
    authenticationRef:
      name: trigger-auth-service-bus-orders
```

```yaml
apiVersion: keda.sh/v1alpha1
kind: TriggerAuthentication
metadata:
  name: trigger-auth-service-bus-orders
spec:
  secretTargetRef:
  - parameter: connection
    name: secrets-order-management
    key: servicebus-order-management-connectionstring
```

https://github.com/kedacore/sample-dotnet-worker-servicebus-queue

Scaling

KEDA

HTTP

```
{
 "name": "http-rule",
  "http": {
   "metadata": {
    "concurrentRequests": 50
   }
  }
}
```

Event-driven

artemis-queue, kafka, aws-cloudwatch, aws-kinesis-stream, aws-sqs-queue, azure-blob, azure-eventhub, azure-servicebus, azure-queue, cron, external, gcp-pubsub, huawei-cloudeye, ibmmq, influxdb, mongodb, mssql, mysql, postgresql, rabbitmq, redis, redis-streams, selenium-grid, solace-event-queue, ..

CPU

```
{
 "name": "cpu-rule",
  "custom": {
   "type": "cpu",
    "metadata": {
     "type": "Utilization",
      "value": "50"
   }
  }
}
```

Memory

```
{
 "name": "mem-rule",
  "custom": {
   "type": "memory",
    "metadata": {
     "type": "AverageValue",
      "value": "512"
   }
  }
}
```

Support for scale to zero and specifying minimum/maximum replicas

Support for specifying minimum/maximum replicas

KEDA – Event Sources and Scalers

Currently available scalers for KEDA

- ActiveMQ Artemis
- Apache Kafka
- AWS CloudWatch
- AWS Kinesis Stream
- AWS SQS Queue
- Azure Blob Storage
- Azure Event Hubs
- Azure Log Analytics
- Azure Monitor
- Azure Pipelines
- Azure Service Bus
- Azure Storage Queue
- CPU
- Cron
- External
- External Push
- Google Cloud Platform Pub/Sub
- Huawei Cloudeye
- IBM MQ
- InfluxDB
- Kubernetes Workload
- Liiklus Topic
- Memory
- Metrics API
- MongoDB
- MSSQL
- MySQL
- NATS Streaming
- OpenStack Metric
- OpenStack Swift
- PostgreSQL
- Prometheus
- RabbitMQ Queue
- Redis Lists
- Redis Lists (supports Redis Cluster)
- Redis Streams
- Redis Streams (supports Redis Cluster)
- Selenium Grid Scaler
- Solace PubSub+ Event Broker

ScaledObject CRD – Deployment, StatefulSets, Custom Resources

```yaml
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  name: {scaled-object-name}
spec:
  scaleTargetRef:
    apiVersion:    {api-version-of-target-resource}  # Optional. Default: apps/v1
    kind:          {kind-of-target-resource}         # Optional. Default: Deployment
    name:          {name-of-target-resource}         # Mandatory. Must be in the same na
    envSourceContainerName: {container-name}         # Optional. Default: .spec.templat
  pollingInterval:  30                               # Optional. Default: 30 seconds
  cooldownPeriod:   300                              # Optional. Default: 300 seconds
  idleReplicaCount: 0                                # Optional. Must be less than minRe
  minReplicaCount:  1                                # Optional. Default: 0
  maxReplicaCount:  100                              # Optional. Default: 100
  fallback:                                          # Optional. Section to specify fal
    failureThreshold: 3                              # Mandatory if fallback section is
    replicas: 6                                      # Mandatory if fallback section is
  advanced:                                          # Optional. Section to specify adv
    restoreToOriginalReplicaCount: true/false        # Optional. Default: false
    horizontalPodAutoscalerConfig:                   # Optional. Section to specify HPA
      behavior:                                      # Optional. Use to modify HPA's sc
        scaleDown:
          stabilizationWindowSeconds: 300
          policies:
          - type: Percent
            value: 100
            periodSeconds: 15
  triggers:
  # {list of triggers to activate scaling of the target resource}
```

ScaledObject CRD – Job

```yaml
apiVersion: keda.sh/v1alpha1
kind: ScaledJob
metadata:
  name: {scaled-job-name}
spec:
  jobTargetRef:
    parallelism: 1                          # [max number of desired pods](https://kub
    completions: 1                          # [desired number of successfully finishe
    activeDeadlineSeconds: 600              #  Specifies the duration in seconds rela
    backoffLimit: 6                         # Specifies the number of retries before
    template:
      # describes the [job template](https://kubernetes.io/docs/concepts/workloads/contr
  pollingInterval: 30                       # Optional. Default: 30 seconds
  successfulJobsHistoryLimit: 5             # Optional. Default: 100. How many complet
  failedJobsHistoryLimit: 5                 # Optional. Default: 100. How many failed
  envSourceContainerName: {container-name}  # Optional. Default: .spec.JobTargetRef.te
  maxReplicaCount: 100                      # Optional. Default: 100
  scalingStrategy:
    strategy: "custom"                      # Optional. Default: default. Which Scalin
    customScalingQueueLengthDeduction: 1    # Optional. A parameter to optimize custo
    customScalingRunningJobPercentage: "0.5"  # Optional. A parameter to optimize custo
    pendingPodConditions:                   # Optional. A parameter to calculate pendi
      - "Ready"
      - "PodScheduled"
      - "AnyOtherCustomPodCondition"
  triggers:
  # {list of triggers to create jobs}
```

Click to add text

Triggers

Service Bus Trigger

```yaml
triggers:
- type: azure-servicebus
  metadata:
    # Required: queueName OR topicName and subscriptionName
    queueName: functions-sbqueue
    # or
    topicName: functions-sbtopic
    subscriptionName: sbtopic-sub1
    # Optional, required when pod identity is used
    namespace: service-bus-namespace
    # Optional, can use TriggerAuthentication as well
    connectionFromEnv: SERVICEBUS_CONNECTIONSTRING_ENV_NAME # This must be a connection
    # Optional
    messageCount: "5" # Optional. Count of messages to trigger scaling on. Default: 5 m
    cloud: Private # Optional. Default: AzurePublicCloud
    endpointSuffix: servicebus.airgap.example # Required when cloud=Private
```

Kafka Trigger

```yaml
triggers:
- type: kafka
  metadata:
    bootstrapServers: kafka.svc:9092
    consumerGroup: my-group
    topic: test-topic
    lagThreshold: '5'
    offsetResetPolicy: latest
    allowIdleConsumers: false
    version: 1.0.0
```

Prometheus Trigger

```yaml
triggers:
- type: prometheus
  metadata:
    # Required
    serverAddress: http://<prometheus-host>:9090
    metricName: http_requests_total
    query: sum(rate(http_requests_total{deployment="my-deployment"}[2m])) # Note: query
    threshold: '100'
```

Trigger Authentication (Env Var, Secret, Pod Identity, Vault)

Pod Identity Auth

```yaml
apiVersion: keda.sh/v1alpha1
kind: TriggerAuthentication
metadata:
  name: azure-servicebus-auth
spec:
  podIdentity:
    provider: azure
```

Secret Auth (connection string)

```yaml
apiVersion: keda.sh/v1alpha1
kind: TriggerAuthentication
metadata:
  name: mongodb-trigger
spec:
  secretTargetRef:
    - parameter: connectionString
      name: mongodb-secret
      key: connect
```

Secret Auth (bearer token)

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: keda-prom-secret
  namespace: default
data:
  bearerToken: "BEARER_TOKEN"
  ca: "CUSTOM_CA_CERT"
---
apiVersion: keda.sh/v1alpha1
kind: TriggerAuthentication
metadata:
  name: keda-prom-creds
  namespace: default
spec:
  secretTargetRef:
    - parameter: bearerToken
      name: keda-prom-secret
      key: bearerToken
      # might be required if you're using a custom CA
    - parameter: ca
      name: keda-prom-secret
      key: ca
```

Dapr primer

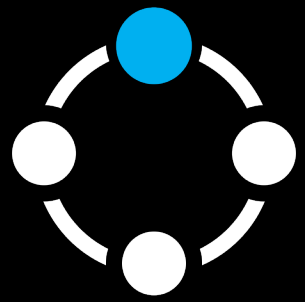
Common microservices requirements

Service to service secure communication **(Enabled with Dapr)**

- 1.1 TLS encryption and mutual TLS authentication ✓
- 1.2 Reliability and retries ✓
- 1.3 Observability and distributed tracing ✓

Independent component lifecycle: versioning and scaling ✓ **(Enabled with revisions and KEDA)**

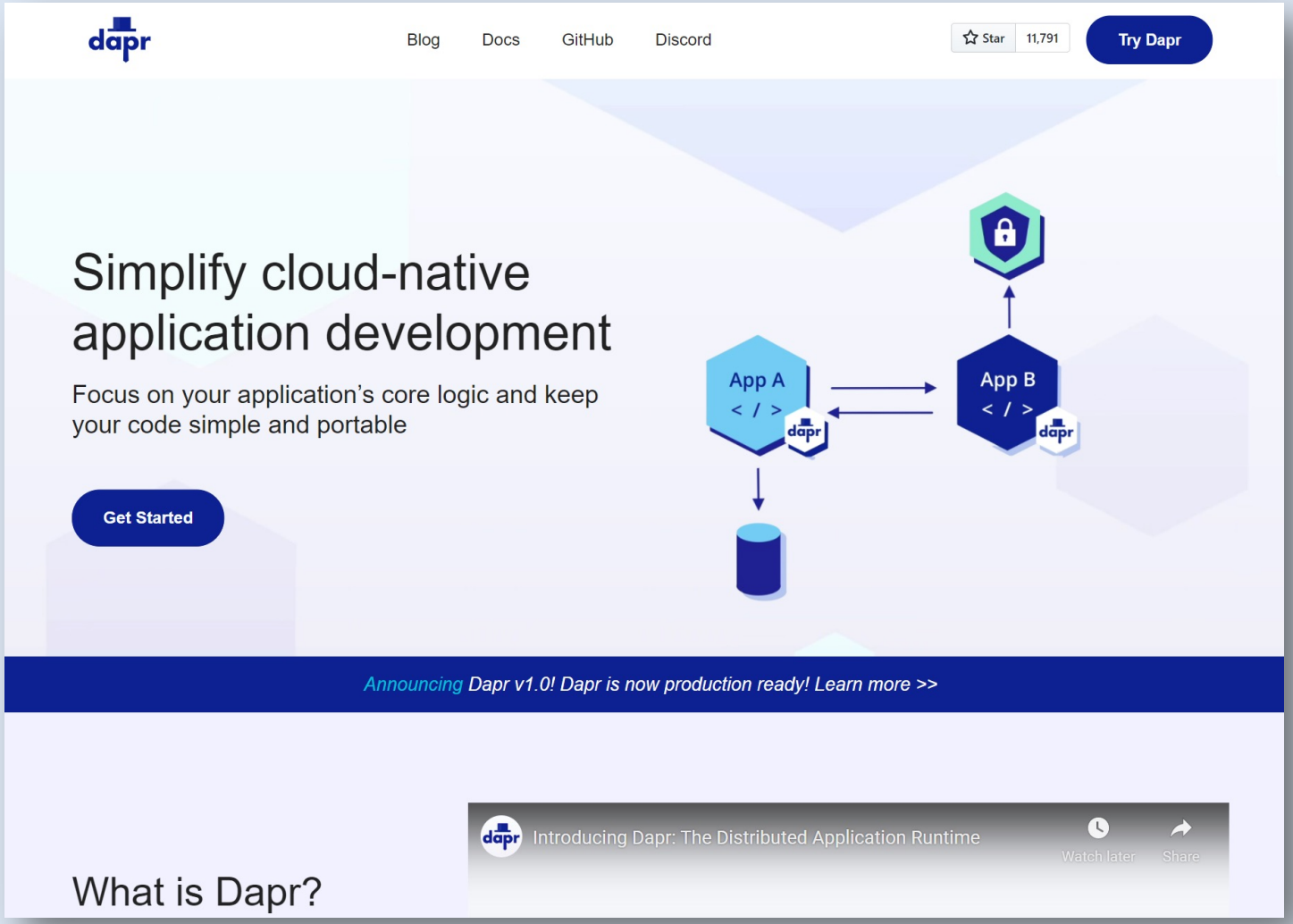
Data encapsulation and governance ✓ **(Enabled with Dapr)**



Distributed Application Runtime

Portable, event-driven, runtime for building distributed applications across cloud and edge

dapr.io



Dapr Goals

Best-practices building blocks

Any language or framework

Consistent, portable, open APIs

Adopt standards

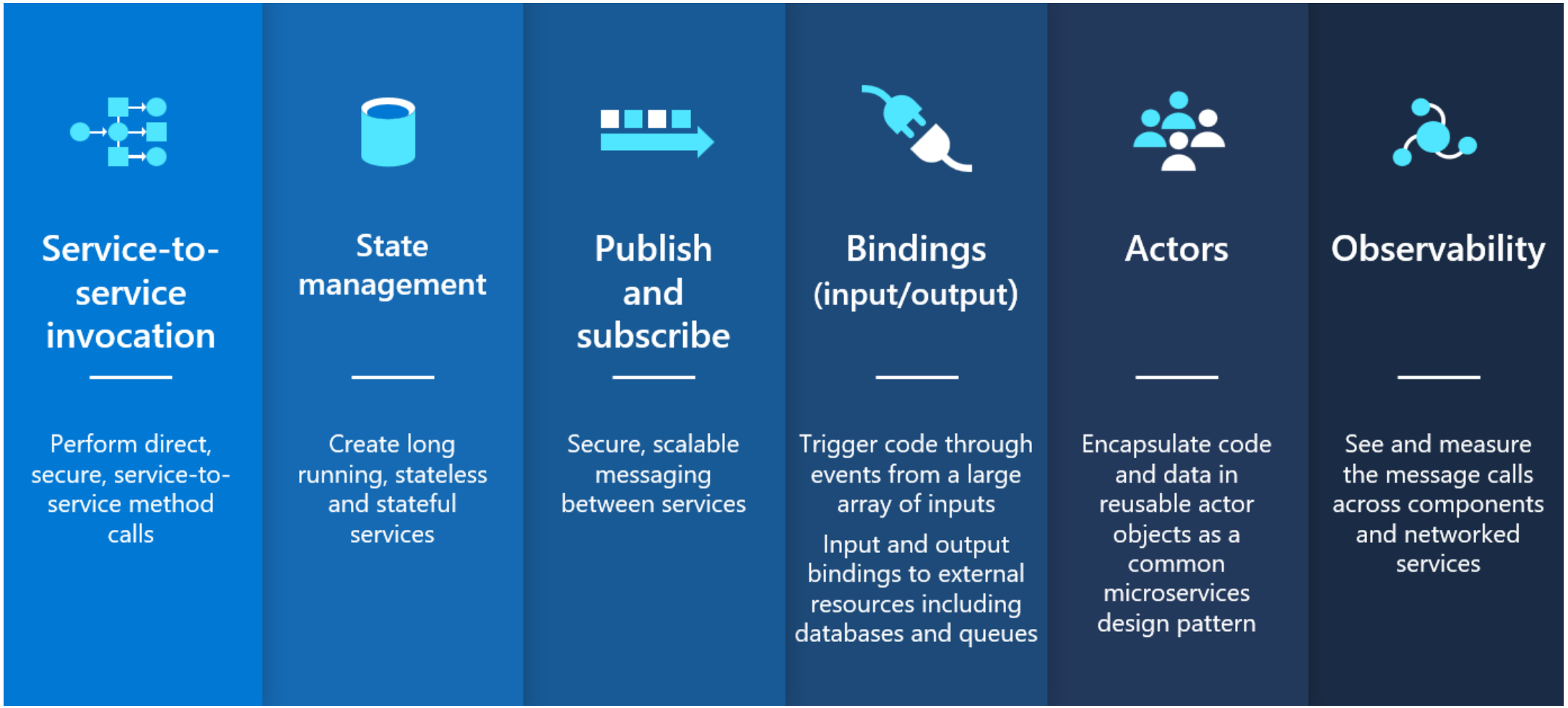
Extensible and pluggable components

Platform agnostic cloud + edge

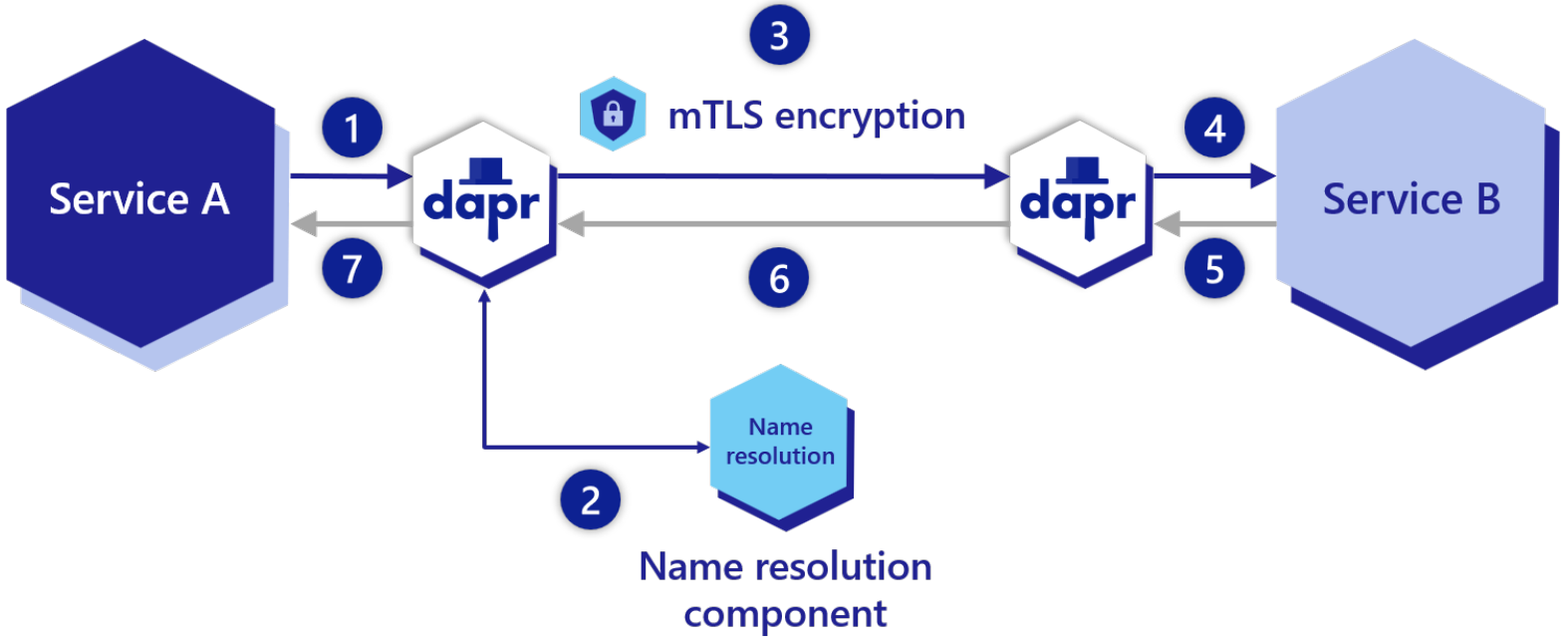
Community driven, vendor neutral

Dapr integration mTLS, service discovery, tracing, etc. (1)

Available Dapr APIs for ContainerApps include:

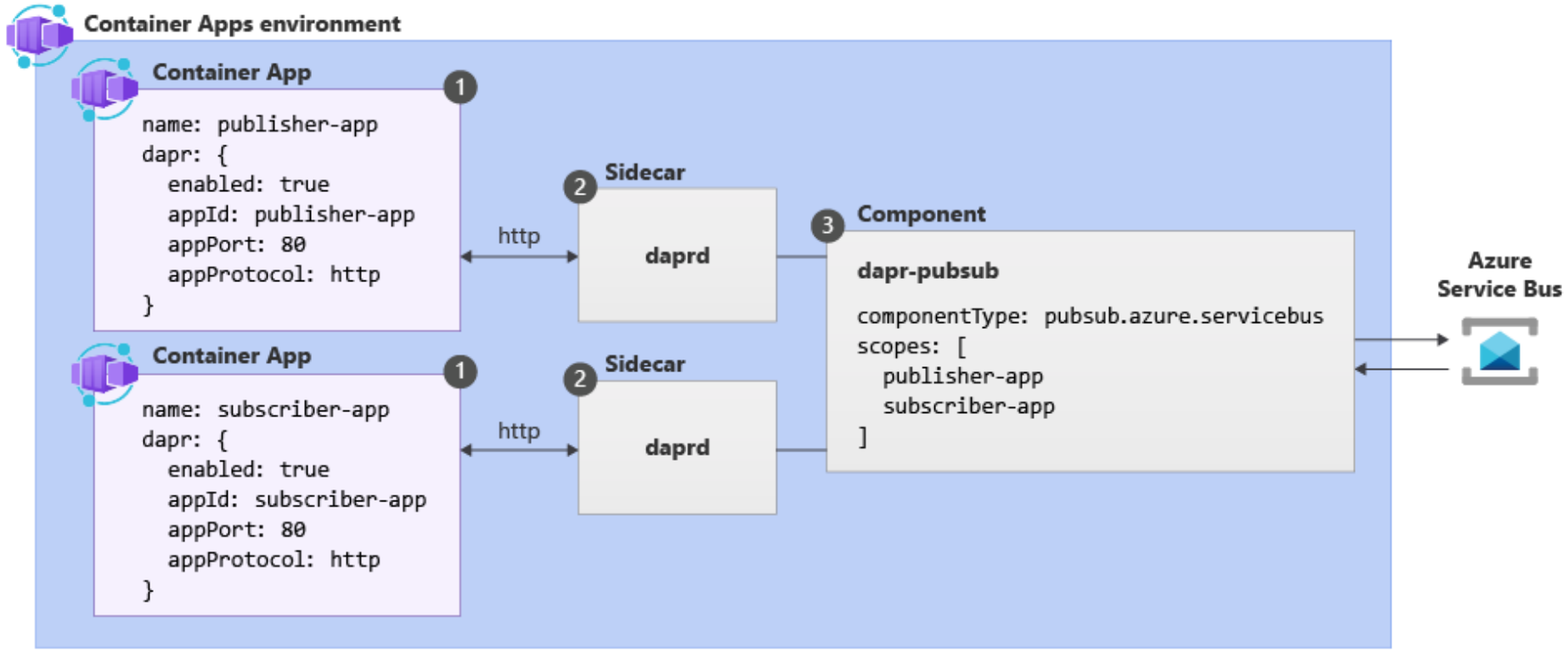


Dapr integration mTLS, service discovery, tracing, etc. (2)



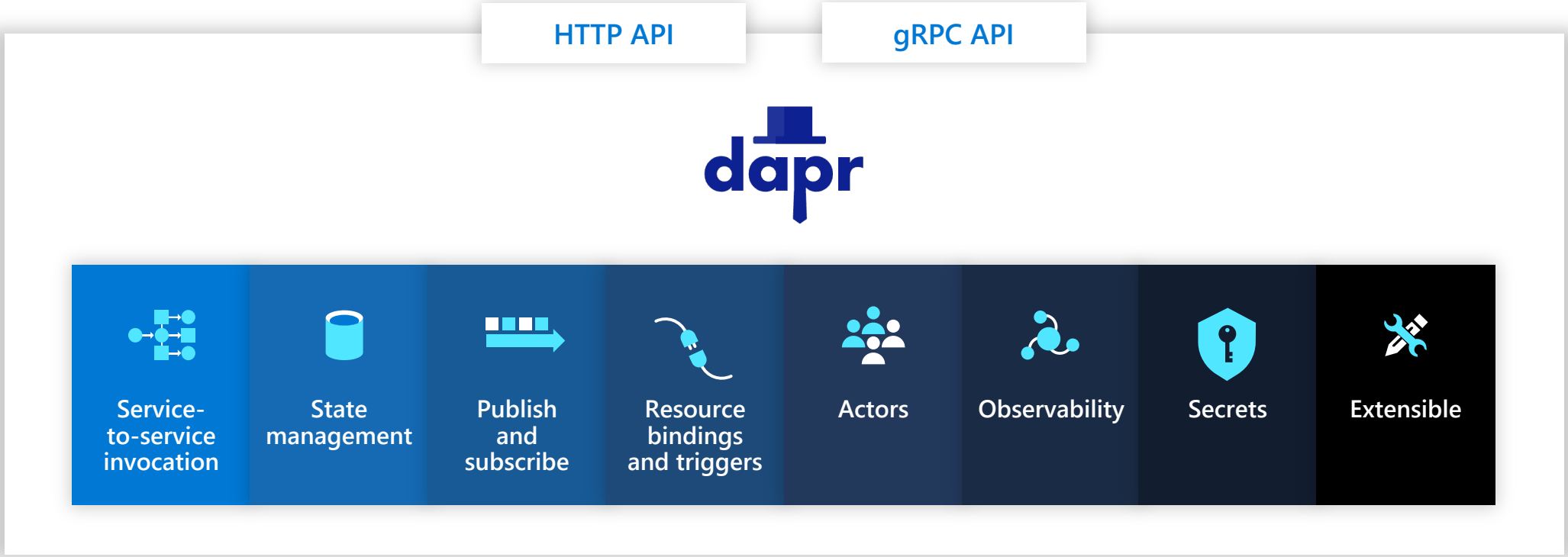
Dapr integration - settings

The following Pub/sub example demonstrates how Dapr works alongside your container app



Label	Dapr settings	Description
1	Container Apps with Dapr enabled	Dapr is enabled at the container app level by configuring Dapr settings. Dapr settings exist at the app-level, meaning they apply across revisions.
2	Dapr sidecar	Fully managed Dapr APIs are exposed to your container app via the Dapr sidecar. These APIs are available through HTTP and gRPC protocols. By default, the sidecar runs on port 3500 in Container Apps.
3	Dapr component	Dapr components can be shared by multiple container apps. Using scopes, the Dapr sidecar will determine which components to load for a given container app at runtime.

Microservice building blocks



Any cloud or edge infrastructure

Application code

Microservices written in

Any code or framework... GO node python .NET Java C++ php

HTTP API | gRPC API

dapr

Service-to-service invocation | State management | Publish and subscribe | Resource bindings and triggers | Actors | Observability | Secrets | Extensible

Hosting infrastructure

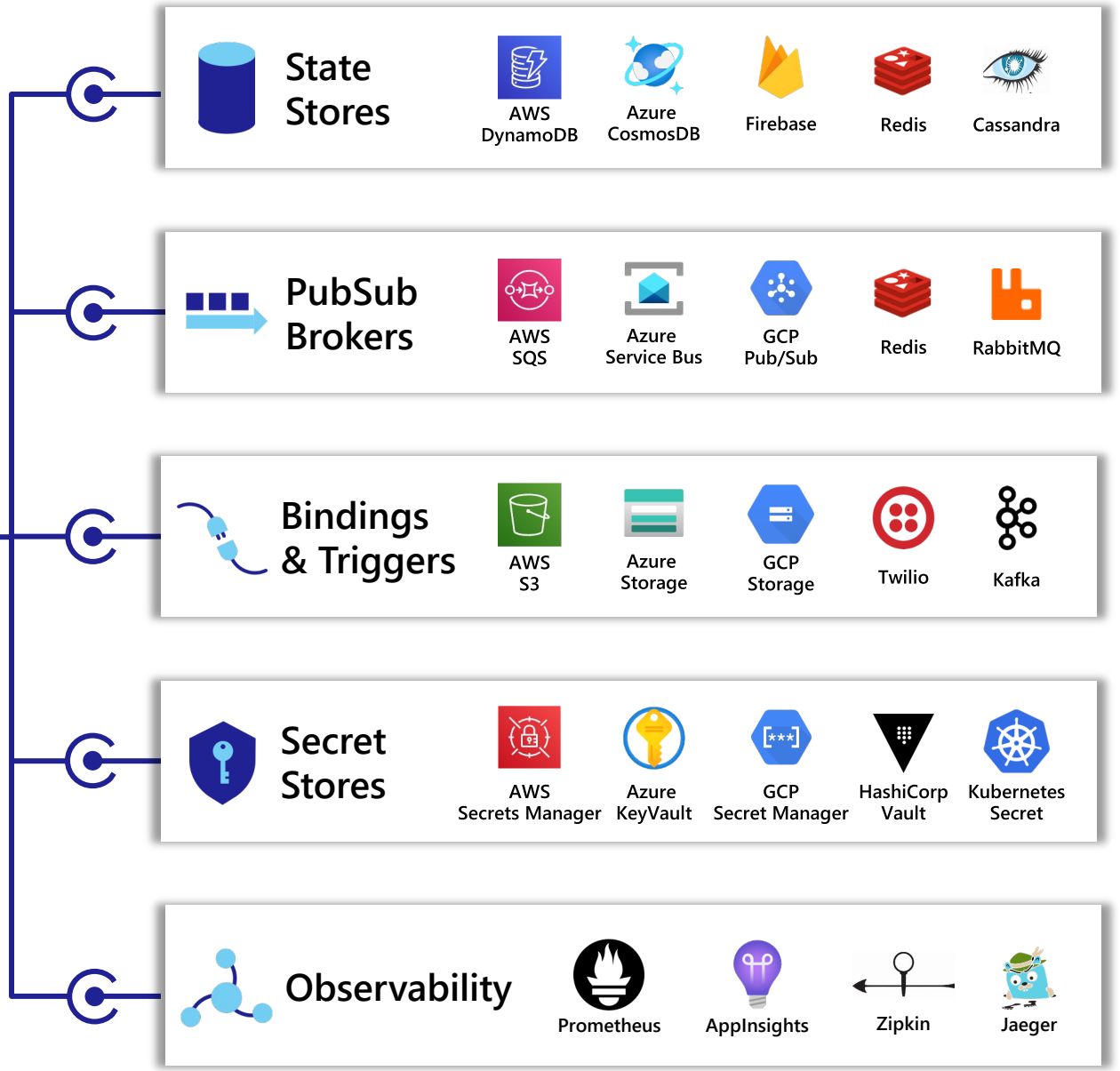
Microsoft Azure | Azure Arc | aws | Google Cloud | Alibaba Cloud | kubernetes | On-Premises

Dapr components

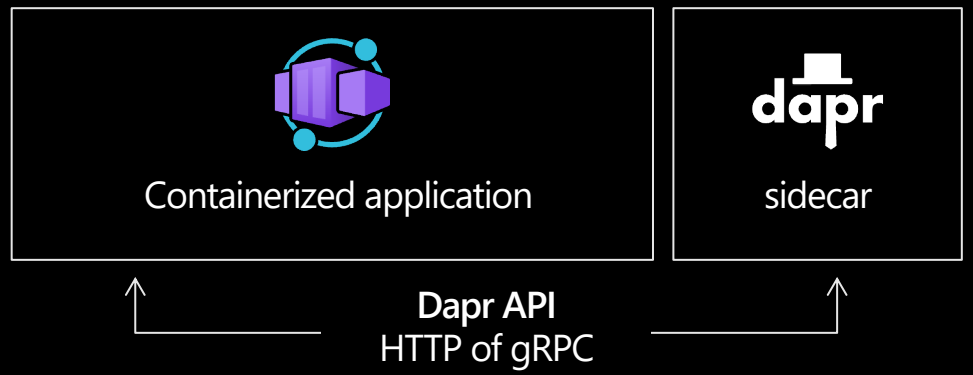
Swappable YAML files with resource connection details

Over 70 components available

Create components for your resource at: github.com/dapr/components-contrib



Fully managed Dapr using the sidecar model



Service-to-service invocation

```
POST http://localhost:3500/v1.0/invoke/cart/method/neworder
```

State management

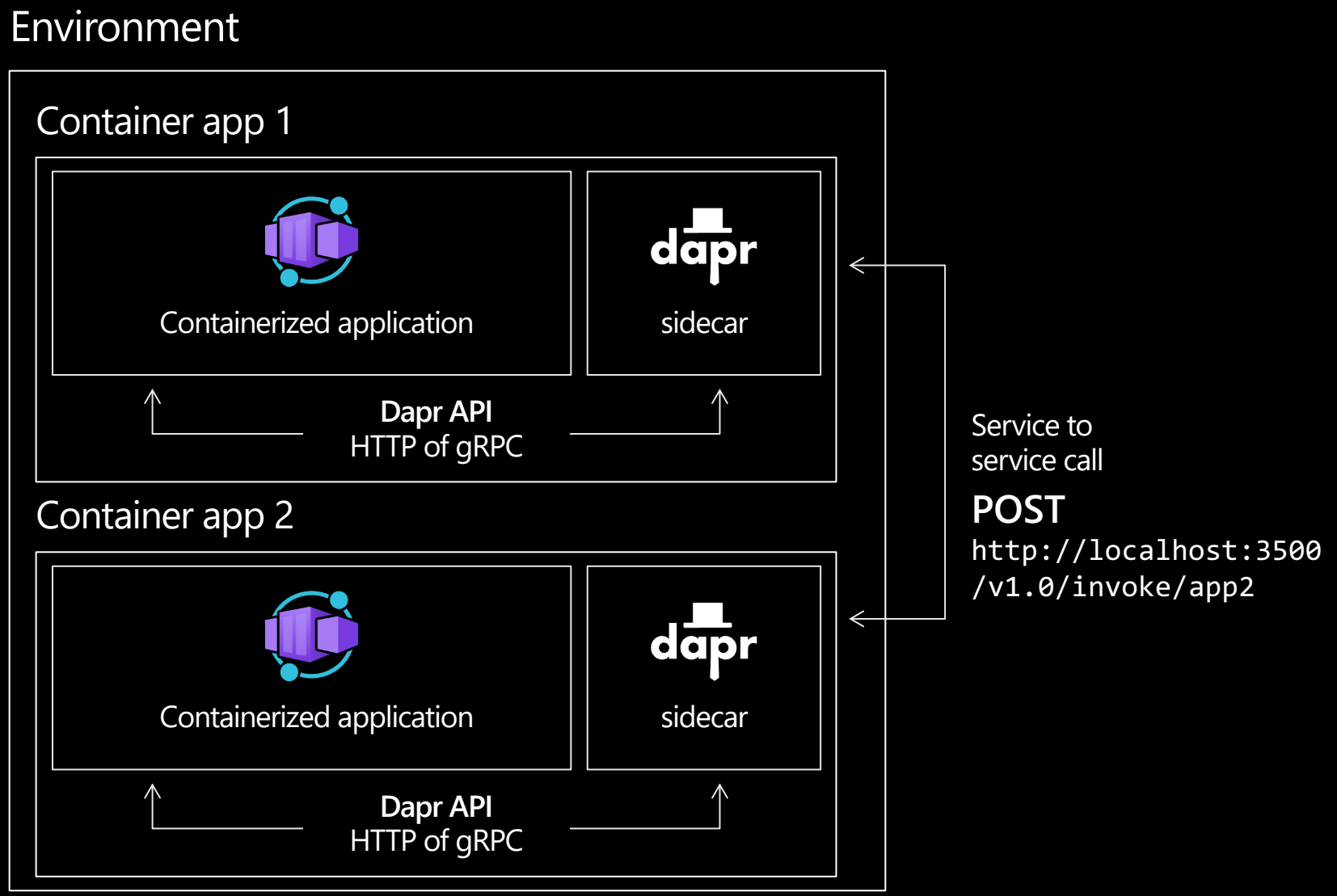
```
GET http://localhost:3500/v1.0/state/inventory/item67
```

Publish and subscribe

```
POST http://localhost:3500/v1.0/publish/shipping/orders
```

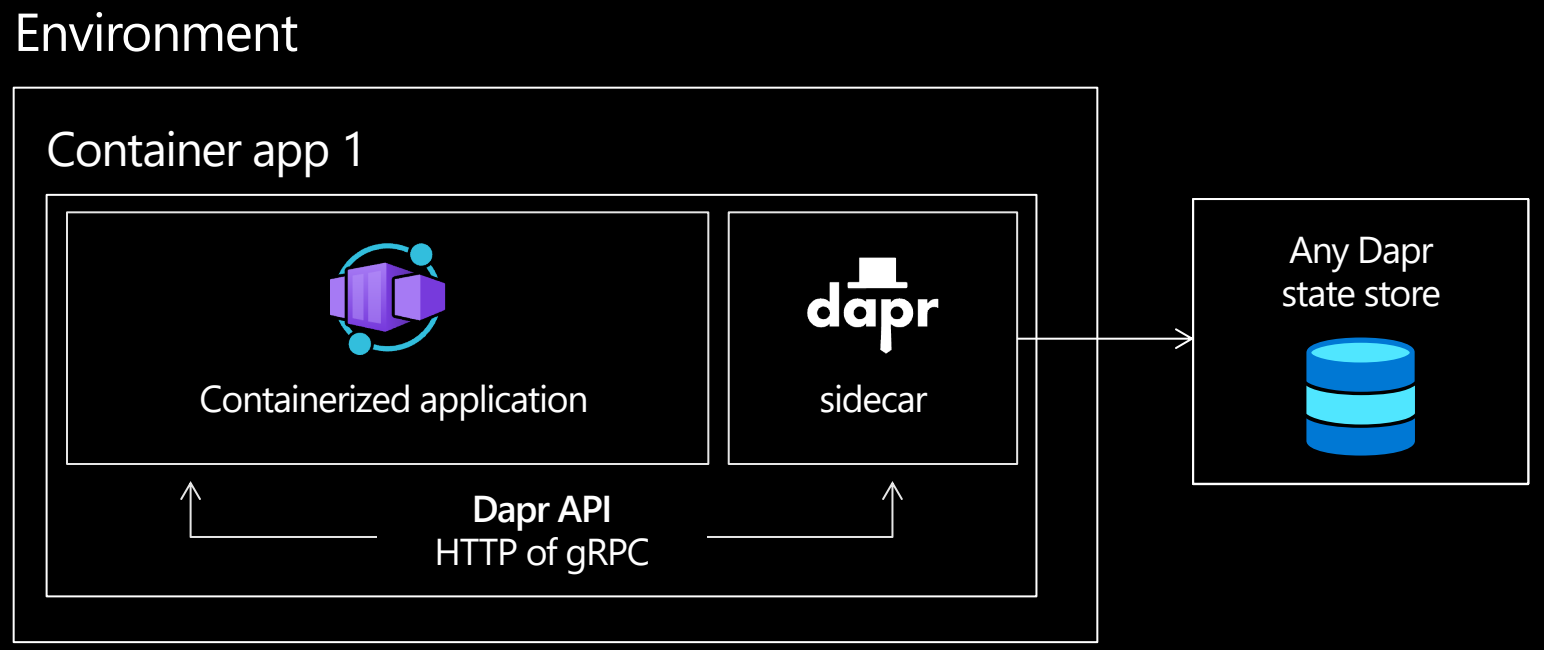
Service to service invocation

Fully managed Dapr APIs provide a rich set of capabilities and productivity gains



State management

Dapr provide apps with state management capabilities for CRUD operations, transactions and more



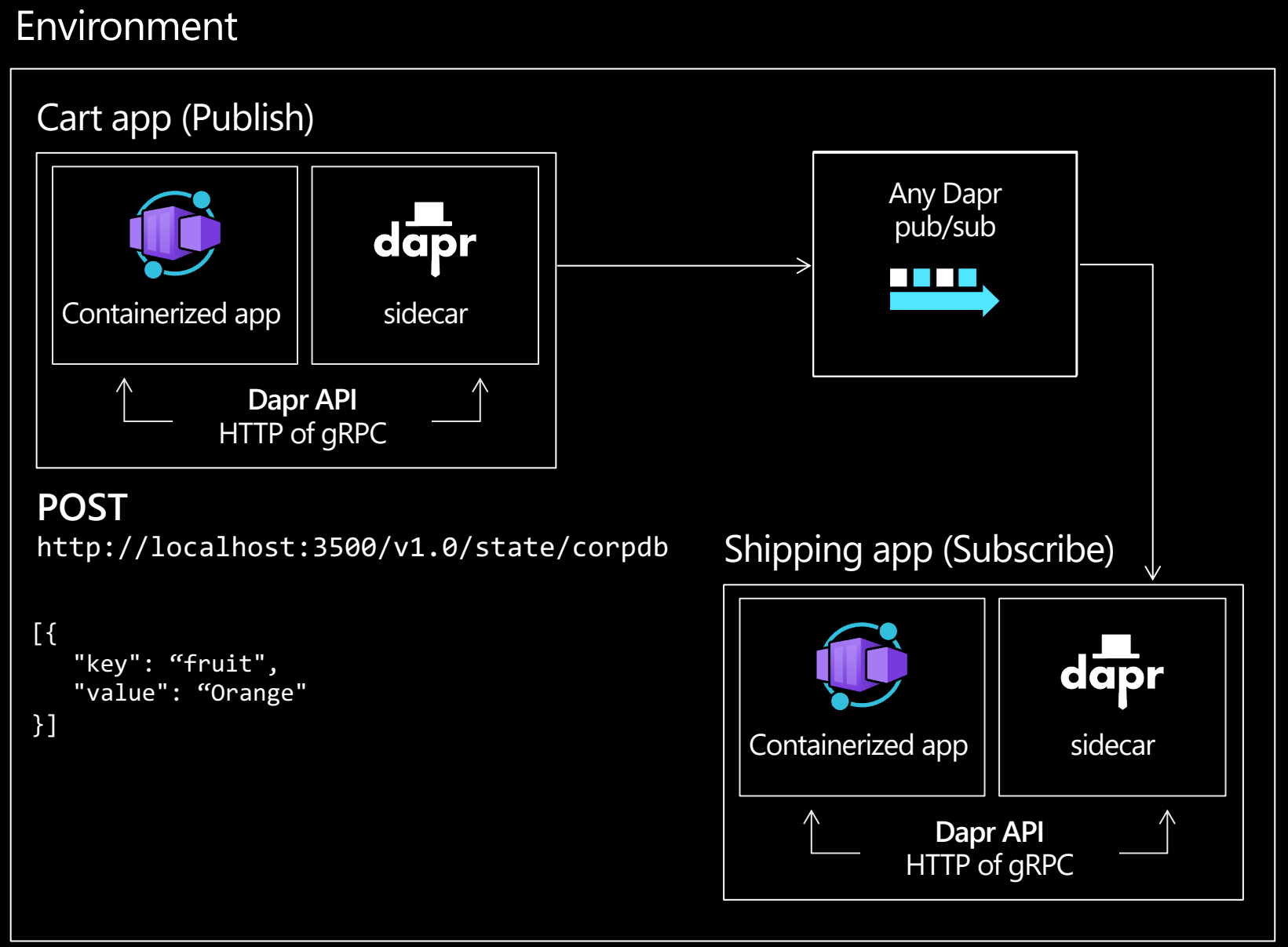
POST

```
http://localhost:3500/v1.0/state/corpdb

[{
   "key": “fruit",
   "value": “Orange"
}]
```

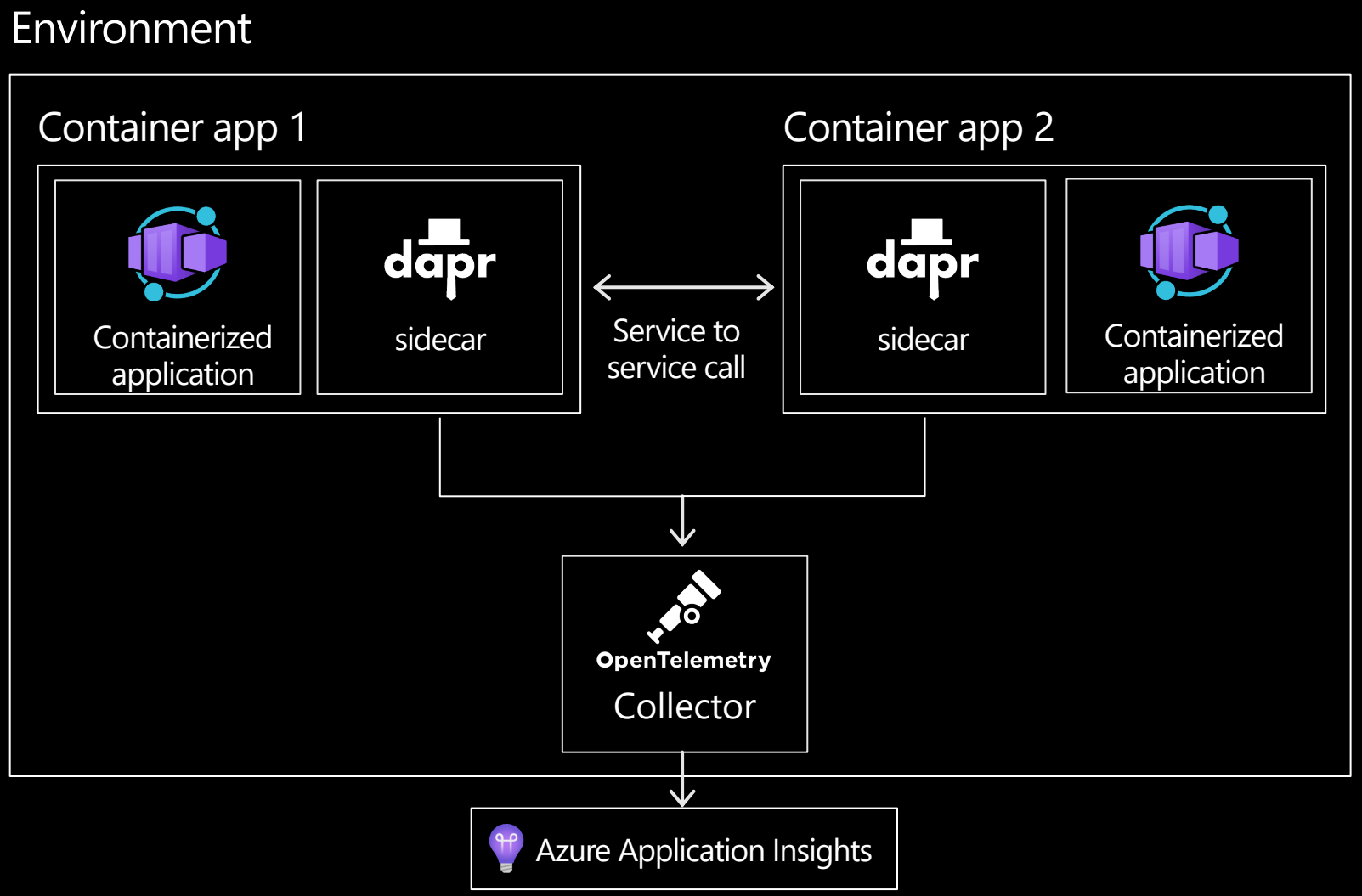
Publish and subscribe

Create event-driven, loosely coupled architectures where producers send events to consumers via topics.



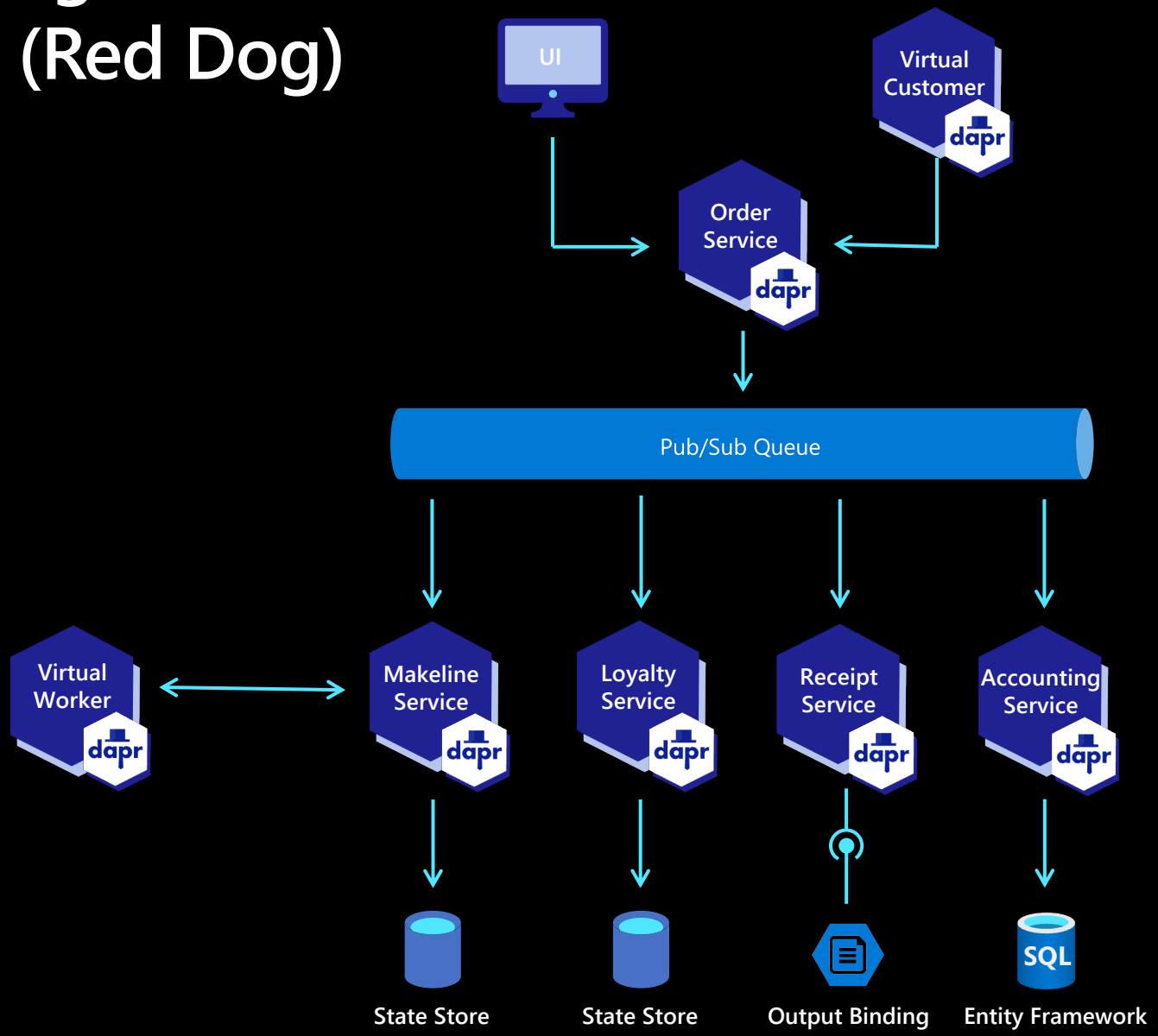
Observability

Intercept traffic and extract tracing, metrics, and logging information. Configure Azure Application Insights for distributed tracing across your services



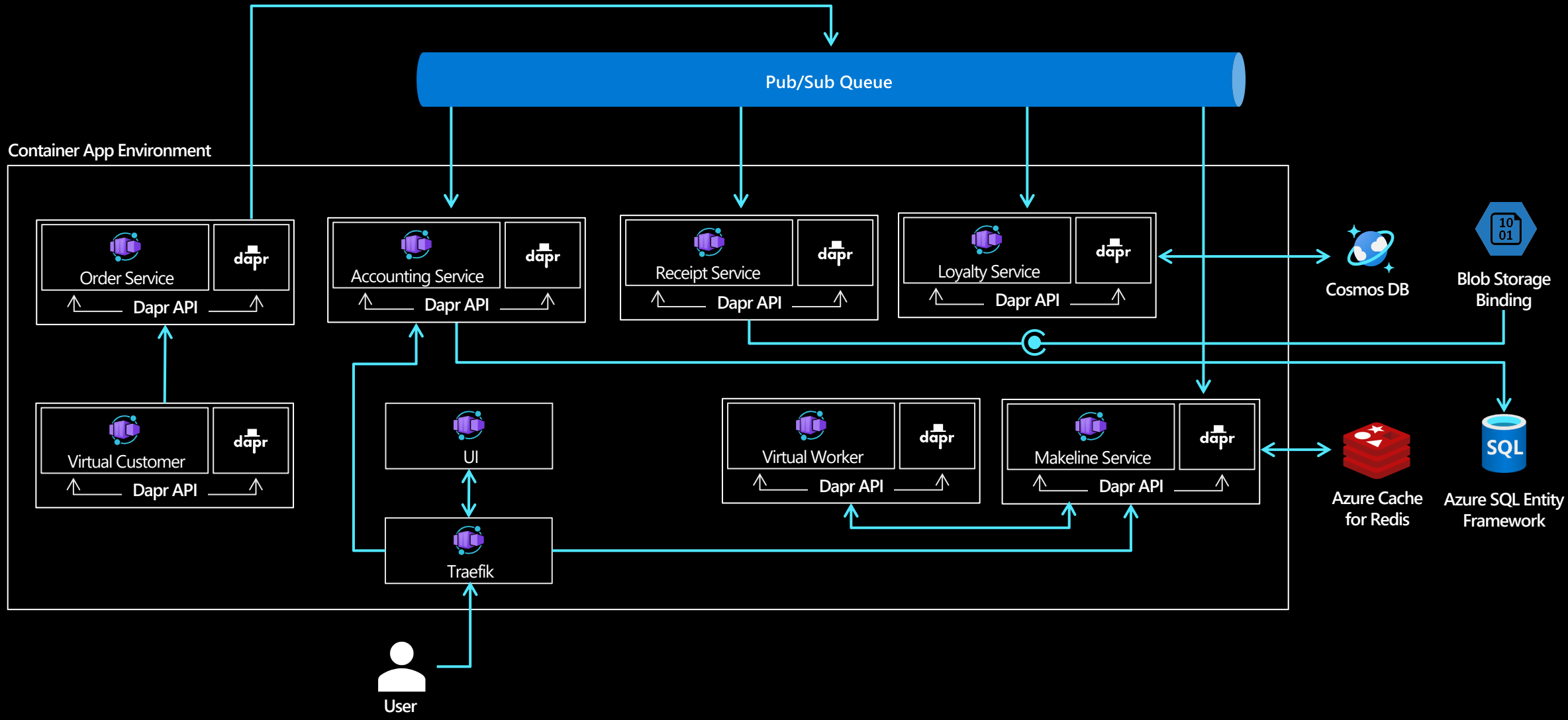
Demo Application

Demo App Logical Architecture (Red Dog)



https://github.com/Azure/reddog-code

Container Apps Architecture



Demo Application - Microservices Details

Service	Ingress	Dapr Component(s)	KEDA Scale Rule(s)
Traefik	External	Dapr not enabled	HTTP
UI	Internal	Dapr not enabled	HTTP
Virtual Customer	None	Service to Service Invocation	N/A
Order Service	Internal	PubSub: Azure Service Bus	HTTP
Accounting Service	Internal	PubSub: Azure Service Bus	Azure Service Bus Subscription Length, HTTP
Receipt Service	Internal	PubSub: Azure Service Bus, Binding: Azure Blob	Azure Service Bus Subscription Length
Loyalty Service	Internal	PubSub: Azure Service Bus, State: Azure Cosmos DB	Azure Service Bus Subscription Length
Makeline Service	Internal	PubSub: Azure Service Bus, State: Azure Redis	Azure Service Bus Subscription Length, HTTP
Virtual Worker	None	Service to Service Invocation, Binding: Cron	N/A

Pricing

Azure Container Apps billing consists of two types of charges:

- **Resource consumption**: The amount of resources allocated to your container app on a per-second basis, billed in vCPU-seconds and GiB-seconds.
- **HTTP requests**: The number of HTTP requests your container app receives.

The following resources are free during each calendar month, per subscription:

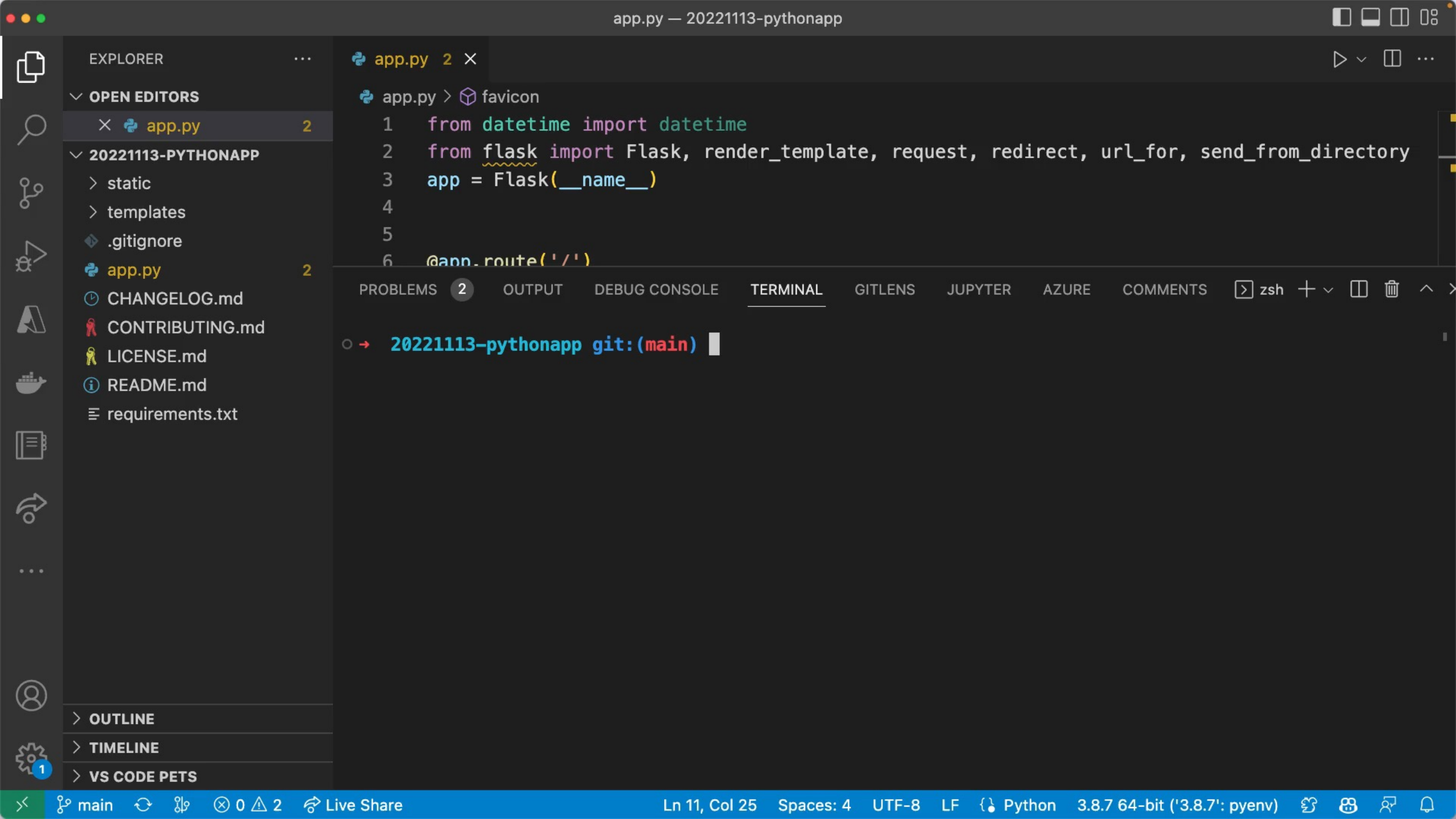
- The first 180,000 vCPU-seconds
- The first 360,000 GiB-seconds
- The first 2 million HTTP requests

When a revision is scaled to zero replicas, no resource consumption charges are incurred.

Billing in Azure Container Apps | Microsoft Learn

Meter	Active Usage Price	Idle Usage Price	Free usage
Requests	$0.40 / million requests	-	2 million requests/month
vCPU (seconds)	$0.086 / vCPU-hr	$0.009 / vCPU-hr	First 180,000 vCPU-s/month
Memory (GiB-seconds)	$0.009 / GiB-hr	$0.009 /GiB-hr	360,000 GiB-s/month

Demo
Azure Container Apps


app.py — 20221113-pythonapp
EXPLORER
OPEN EDITORS
app.py 2
20221113-PYTHONAPP
static
templates
.gitignore
app.py 2
CHANGELOG.md
CONTRIBUTING.md
LICENSE.md
README.md
requirements.txt
OUTLINE
TIMELINE
VS CODE PETS
app.py 2
app.py > favicon
1 from datetime import datetime
2 from flask import Flask, render_template, request, redirect, url_for, send_from_directory
3 app = Flask(__name__)
4
5
6 @app.route('/')
PROBLEMS 2
OUTPUT
DEBUG CONSOLE
TERMINAL
GITLENS
JUPYTER
AZURE
COMMENTS
zsh
20221113-pythonapp git:(main)
main
0 2
Live Share
Ln 11, Col 25
Spaces: 4
UTF-8
LF
Python
3.8.7 64-bit ('3.8.7': pyenv)