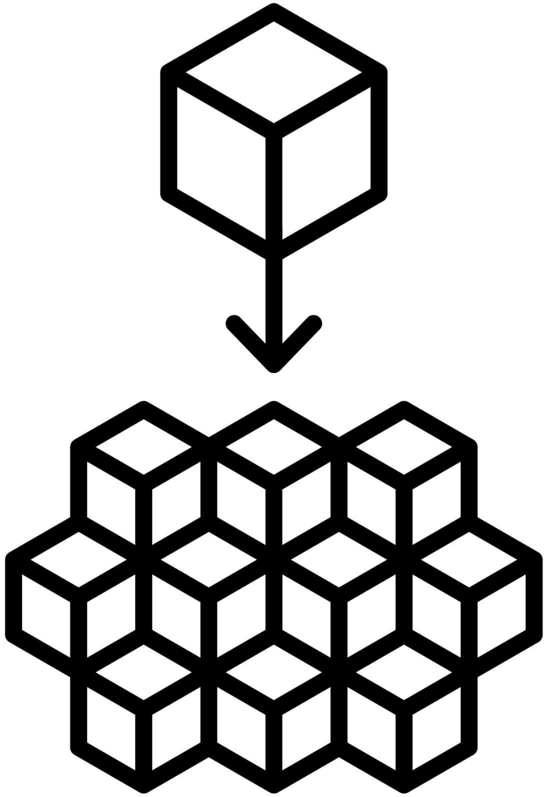
Migrate to Microservices with Dapr

Ronald Harmsen

<DevSum>



Microsoft Azure

kubernetes

Microsoft Azure
Service Fabric

Ronald Harmsen

@ronaldharmsen

ronald@nforza.nl

State of microservice developers

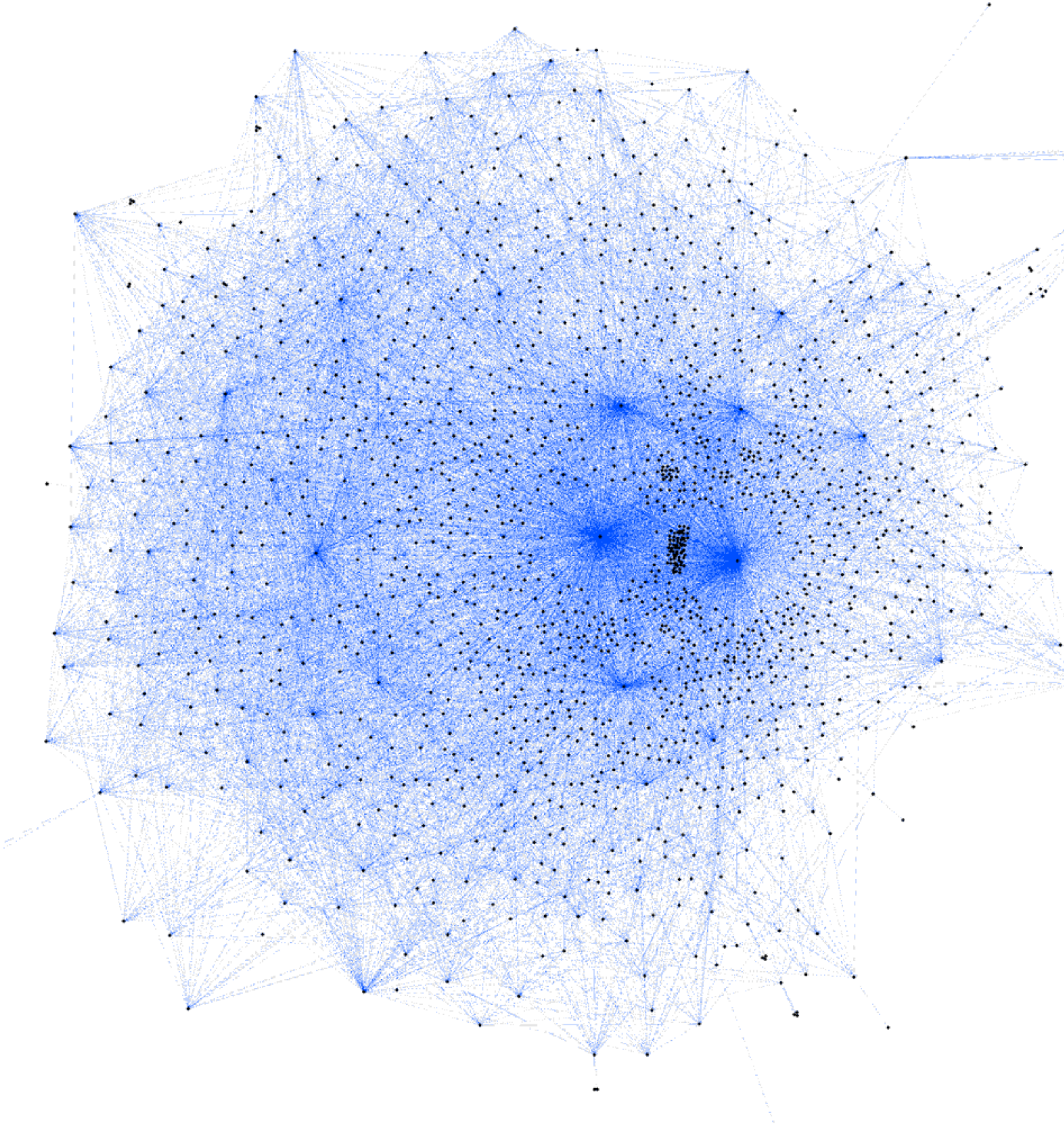
- Being asked to develop resilient, scalable, microservice-based apps
- Write code in many languages
- Leverage existing code
- Functions and Actors are powerful programming models

dapr

Event-driven, portable runtime for building microservice apps on cloud and edge.

Microservices & Kubernetes

- Complexity of setup
  - Learning curve for developers
  - Infrastructure config
- Every K8S cluster is different
  - Ingress controllers
  - Autoscalers
  - Service meshes
  - Logging & Tracing

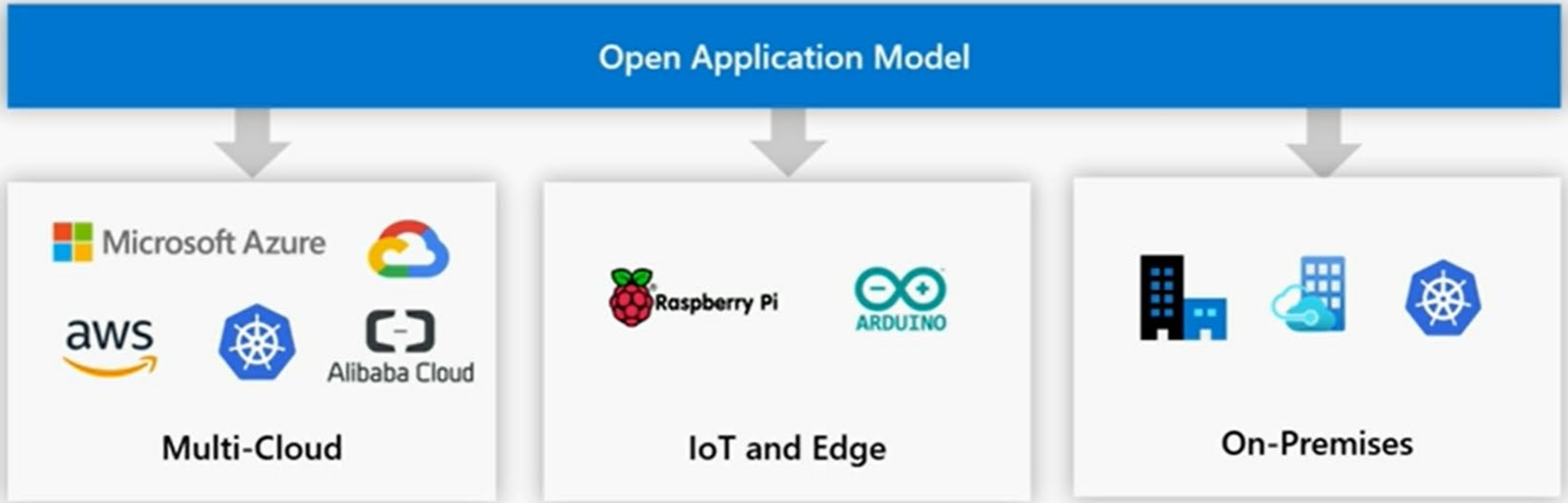


Cloud + Edge

A standard, platform-agnostic application definition for any platform in any environment.

Consistent application modeling for small devices, Kubernetes on prem or cloud, and fully-managed cloud environments.

Extendable by design to leverage the native APIs, tools, and unique features of platforms that users know and love



Is Dapr a Service Mesh?

dapr

Istio

linkerd

Introducing Dapr

A portable, event-driven, serverless runtime for building distributed applications across cloud and edge

Sidecar Architecture

Developer first, standard APIs used from any programming language or framework

Microservice Building Blocks

Make it easy for developers to create microservice applications without being an expert in distributed systems, including migrating existing code

Cloud + Edge

Runs on multiple environments for cloud, on-prem, and small-edge including any Kubernetes

Microservice Building Blocks

State Management

Create long running, stateless and stateful services

Service Invocation & Fault Handling

Perform direct, secure, service-to-service method calls

Resource Bindings

Trigger code through events from a large array of input and output bindings to external resources including databases and queues

Publish & Subscribe

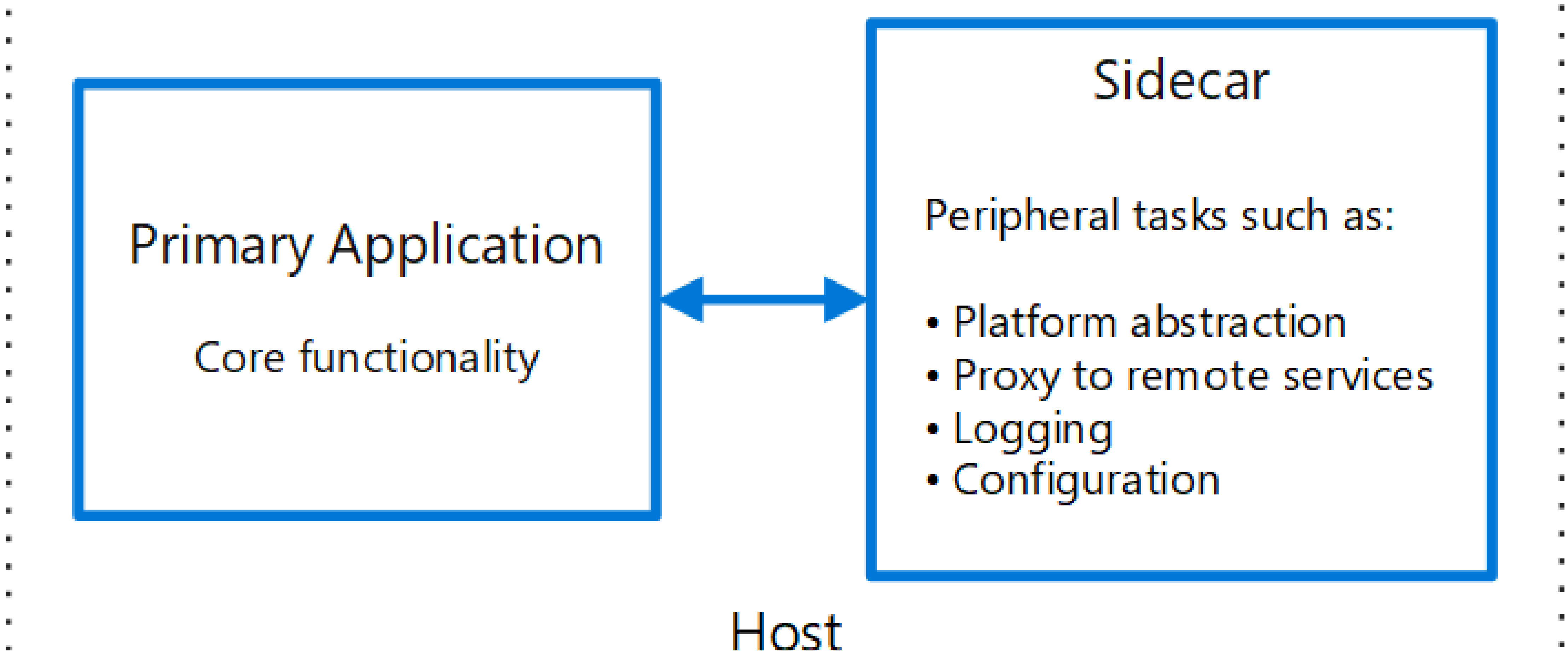
Secure, scalable messaging between services

Actors

Encapsulate code and data in reusable actor objects as a common microservices design pattern

Distributed Tracing & Diagnostics

See and measure the message calls across components and networked services



Primary Application

Core functionality

Sidecar

Peripheral tasks such as:

- Platform abstraction
- Proxy to remote services
- Logging
- Configuration

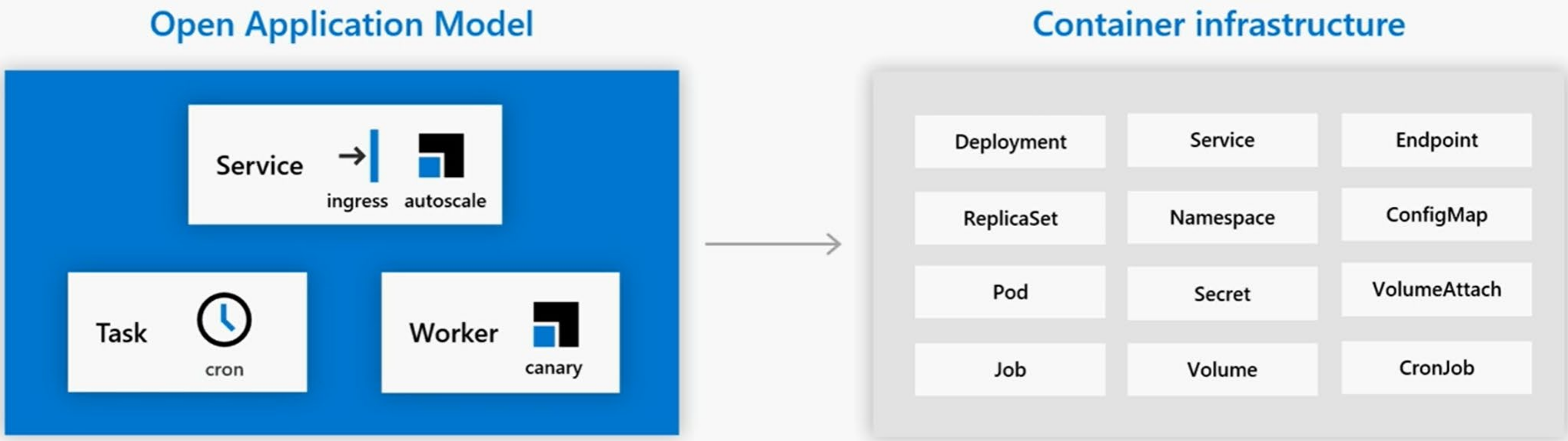
Host

Application focused

Describes application components and operations as first-class concepts without having to stitch together individual container primitives

Flexible application modeling supports a wide range of application architectures

Small and simple applications are easy, large and complex applications are manageable



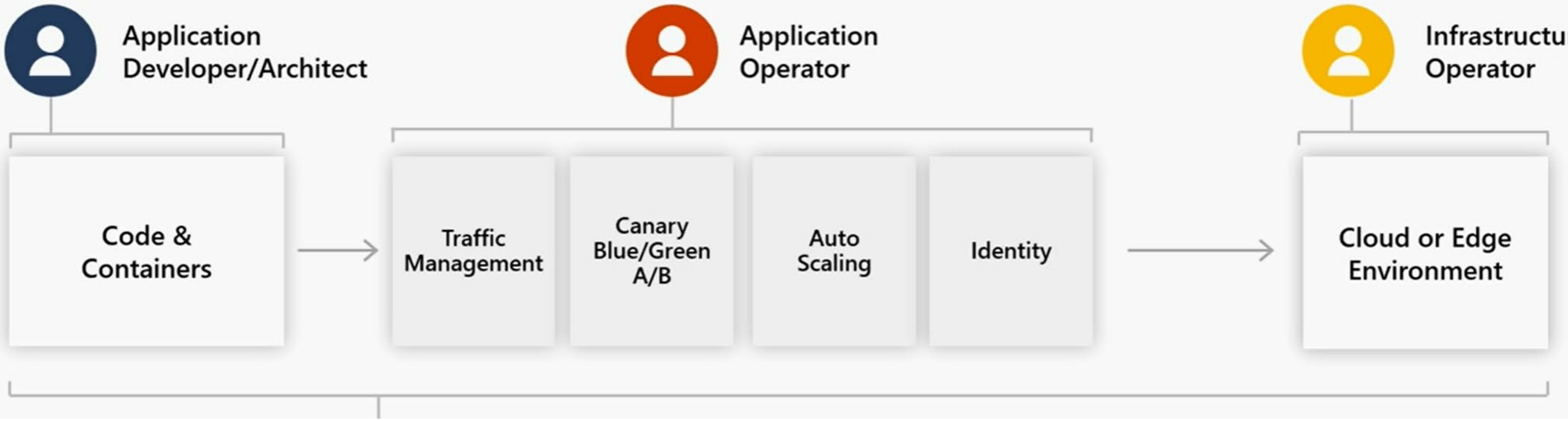
Deployment	Service	Endpoint
ReplicaSet	Namespace	ConfigMap
Pod	Secret	VolumeAttach
Job	Volume	CronJob

Separation of concerns

Allows application developers to focus on their code in a platform-neutral setting to deliver business value

Application operators use powerful and extensible operational traits consistently across platforms and environments

Infrastructure operators can configure their environments to satisfy any unique operating requirements



Application Developer/Architect

Application Operator

Infrastructure Operator

Code & Containers

Traffic Management

Canary Blue/Green A/B

Auto Scaling

Identity

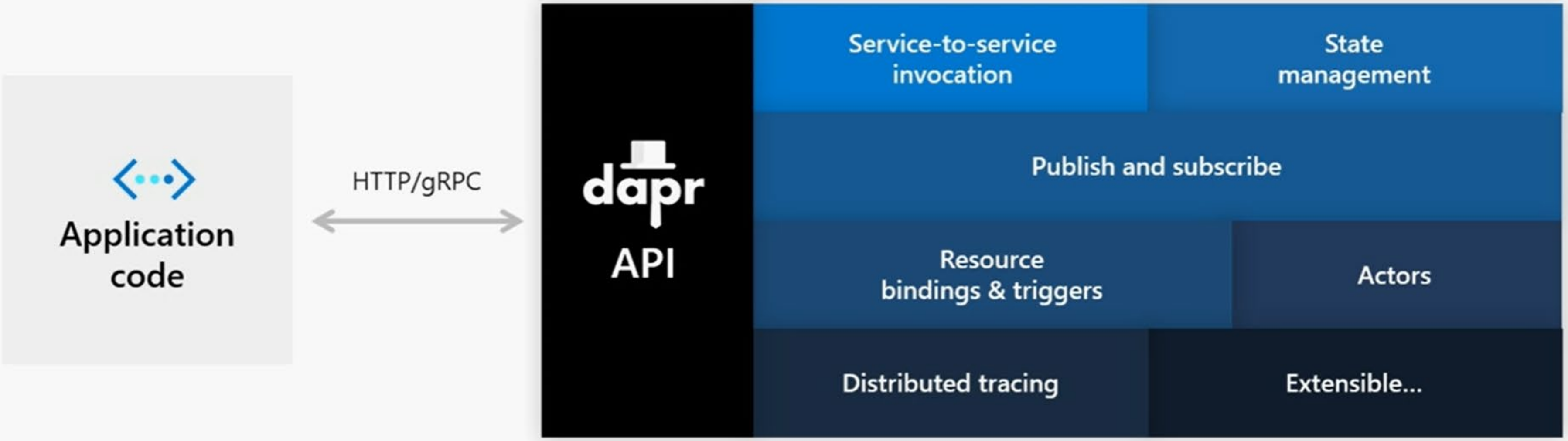
Cloud or Edge Environment

Sidecar architecture

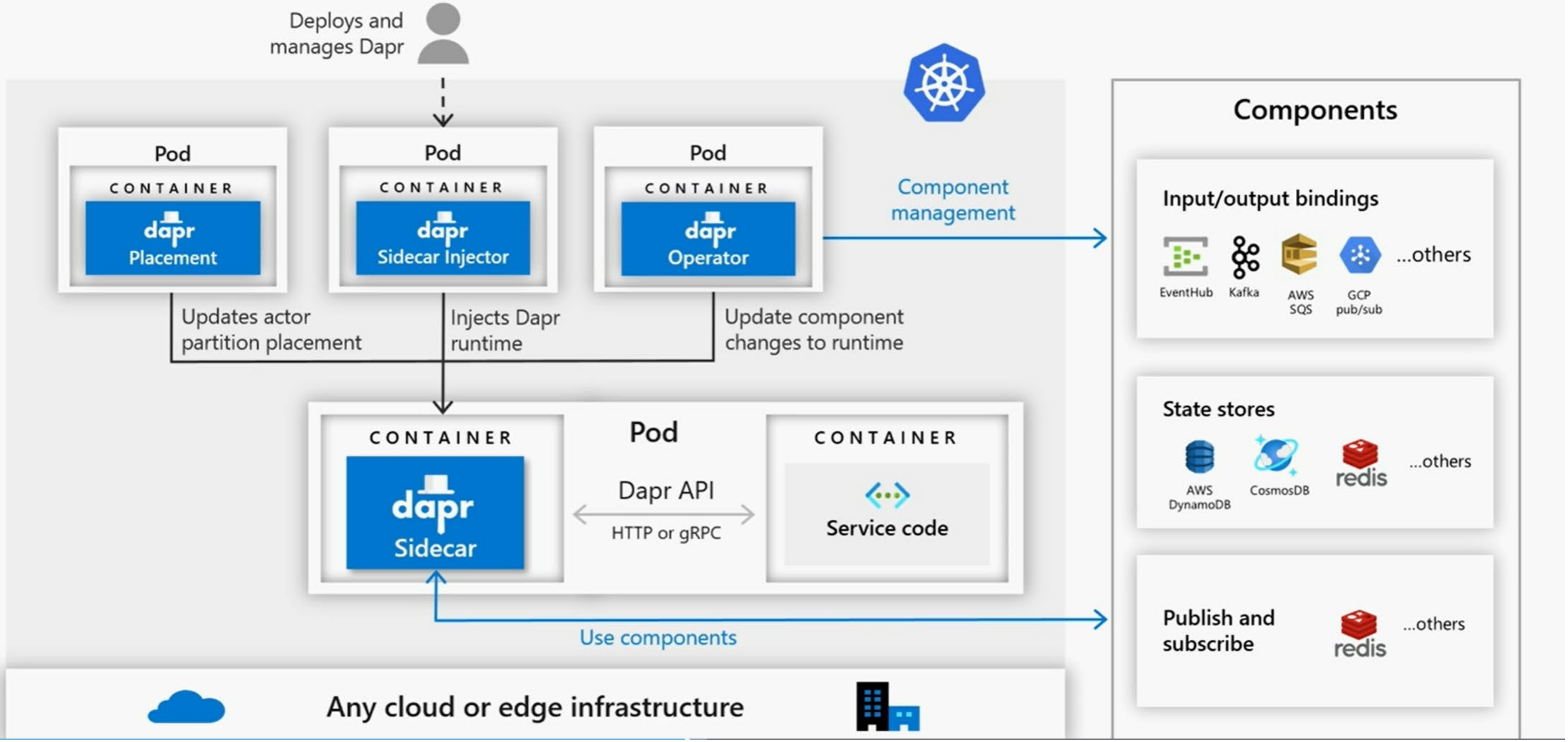
Standard APIs accessed over http/gRPC protocols from user service code

`e.g. http://localhost:3500/v1.0/invoke/myapp/method/neworder`

Dapr runs as local “side-car library” dynamically loaded at runtime for each service



Dapr Kubernetes-hosted



Deploys and manages Dapr

Pod

CONTAINER

dapr Placement

Pod

CONTAINER

dapr Sidecar Injector

Pod

CONTAINER

dapr Operator

Component management

Updates actor partition placement

Injects Dapr runtime

Update component changes to runtime

Pod

CONTAINER

dapr Sidecar

Dapr API

HTTP or gRPC

CONTAINER

Service code

Use components

Any cloud or edge infrastructure

Components

Input/output bindings

EventHub

Kafka

AWS SQS

GCP pub/sub

...others

State stores

AWS DynamoDB

CosmosDB

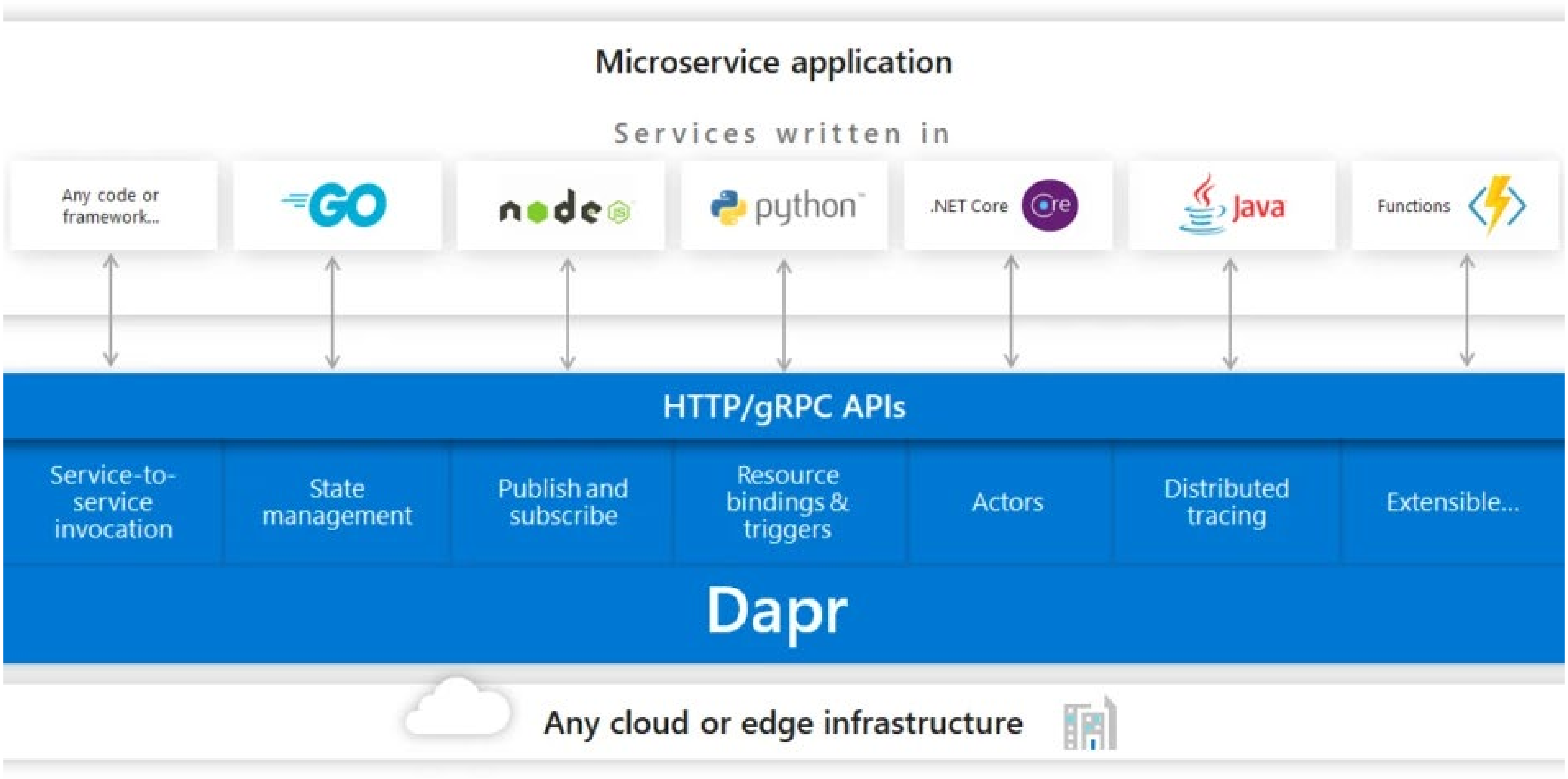
redis

...others

Publish and subscribe

redis

...others



Microservice application

Services written in

Any code or framework...

GO

node

python

.NET Core

Java

Functions

HTTP/gRPC APIs

Service-to-service invocation

State management

Publish and subscribe

Resource bindings & triggers

Actors

Distributed tracing

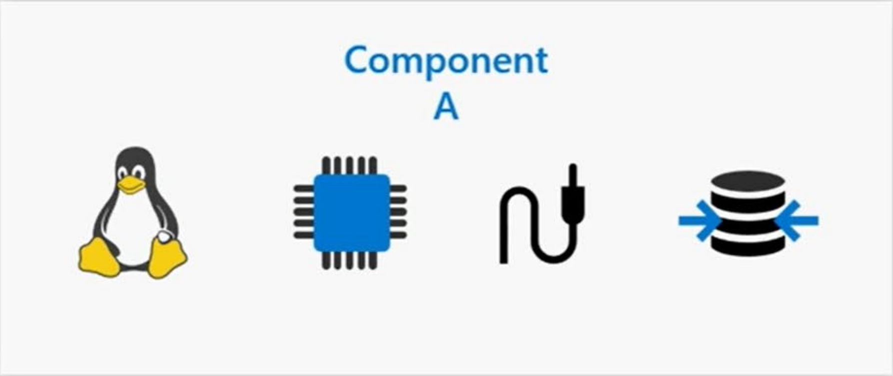
Extensible...

Dapr

Any cloud or edge infrastructure

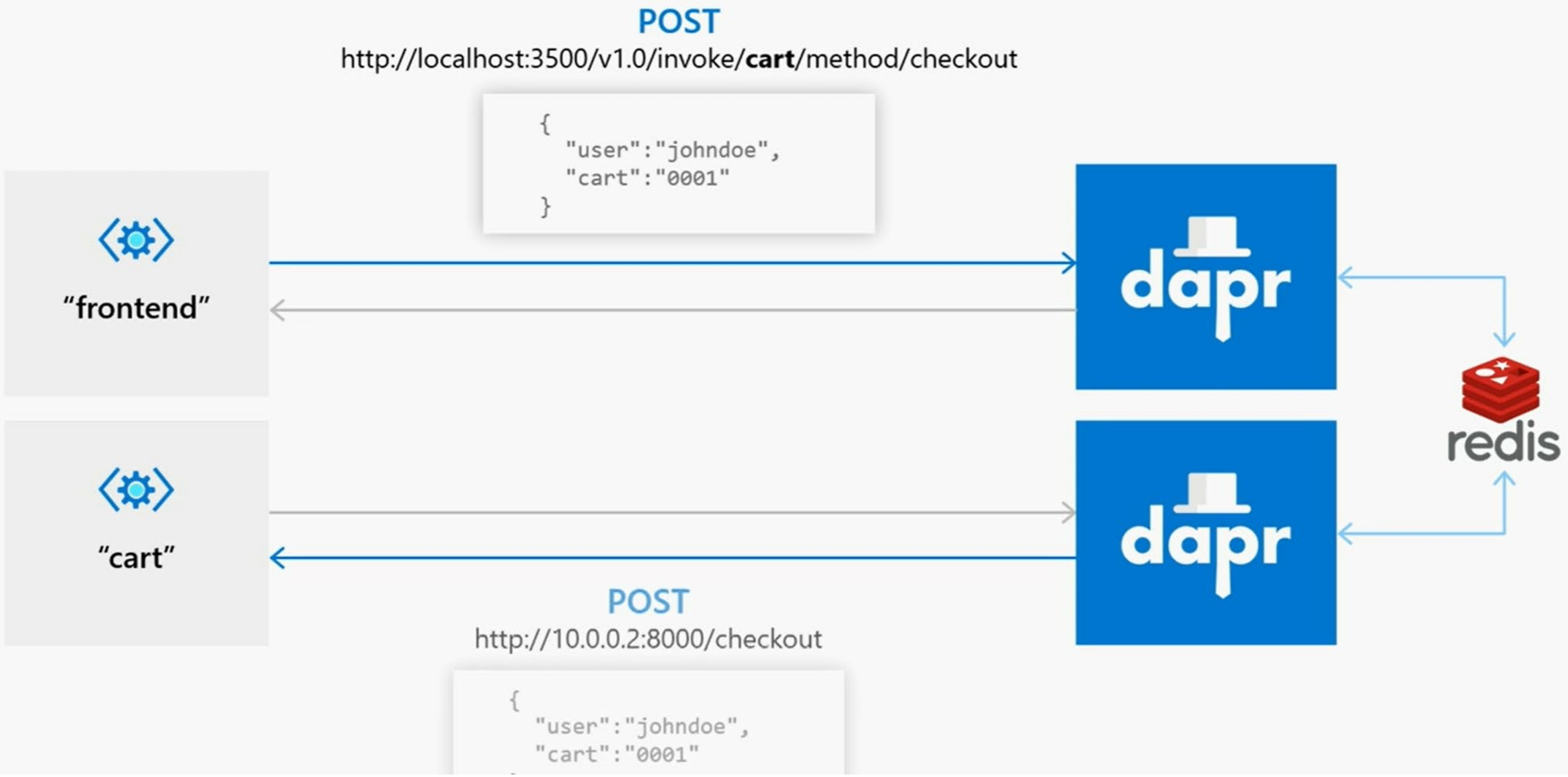
Component

Where developers declare the operational characteristics of the code they deliver *in infrastructure neutral terms.*



```
apiVersion: core.oam.dev/v1alpha1
kind: Component
metadata:
  name: oamfrontend
  version: "1.0.0"
  description: Simple OAM app
spec:
  workloadType: core.oam.dev/v1alpha1.Server
  os: linux
  arch: amd64
  parameters:
    - name: oam_texture
      type: string
      required: true
      default: texture.jpg
  containers:
    - name: frontend
      image: ignite2019/oamhwfrontend:latest
      env:
        - name: OAM_TEXTURE
          value: texture.jpg
          fromParam: oam_texture
      ports:
        - containerPort: 8001
          name: http
          protocol: TCP
```

Service Invocation



POST
http://localhost:3500/v1.0/invoke/**cart**/method/checkout

```
{
  "user":"johndoe",
  "cart":"0001"
}
```

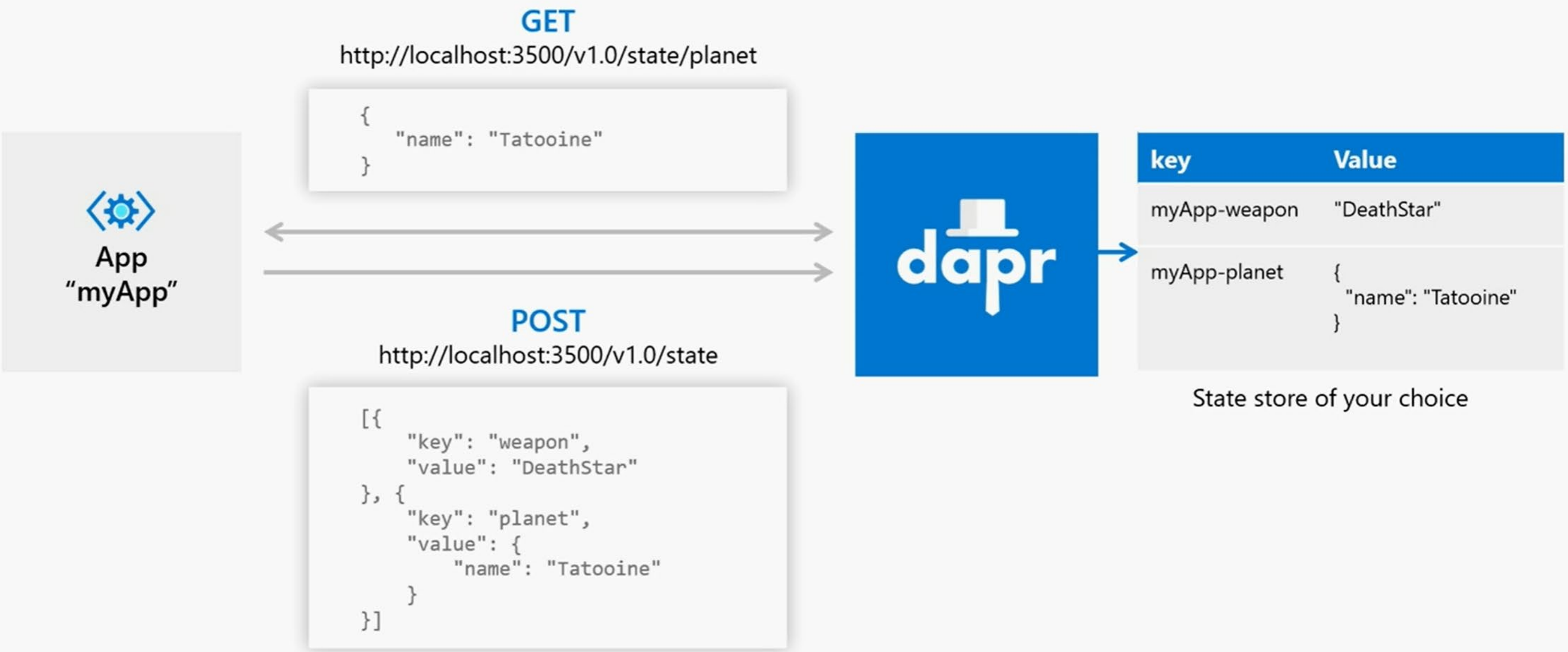
POST
http://10.0.0.2:8000/checkout

```
{
  "user":"johndoe",
  "cart":"0001"
```

Components: State Store

- AWS DynamoDB
- Azure CosmosDB
- Azure Table Storage
- Cassandra
- Cloud Firestore (Datastore mode)
- CloudState
- Etcd
- HashiCorp Consul
- Hazelcast
- Memcached
- MongoDB
- Redis
- SQL Server
- Zookeeper
- Cloud Firestore (Datastore mode)
- Couchbase

State management



GET

http://localhost:3500/v1.0/state/planet

```
{
    "name": "Tatooine"
}
```

POST

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

```
[{
     "key": "weapon",
     "value": "DeathStar"
}, {
     "key": "planet",
     "value": {
         "name": "Tatooine"
     }
}]
```

key	Value
myApp-weapon	"DeathStar"

myApp-planet	{
	"name": "Tatooine"
	}

State store of your choice

Demo

Service invocation and storing state

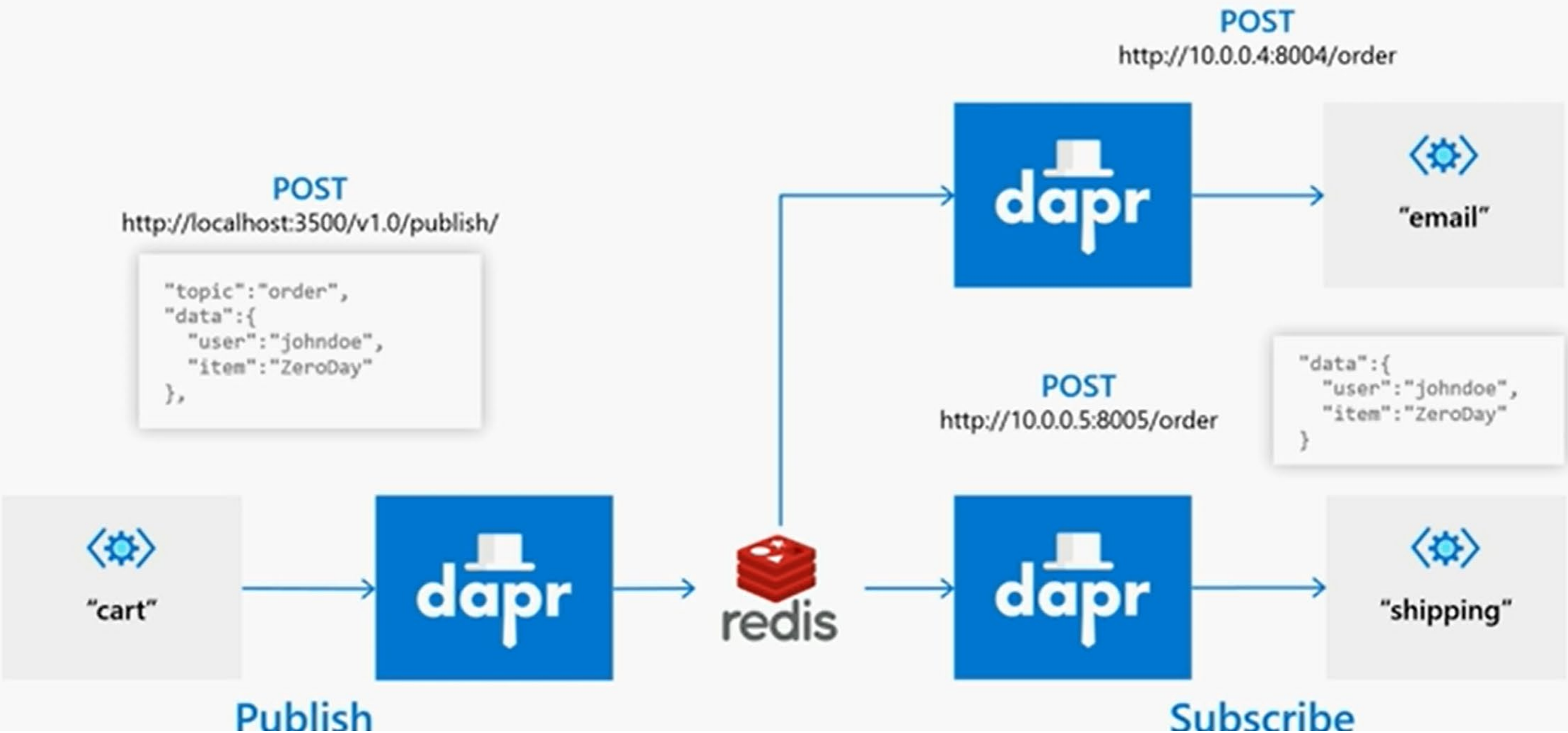
Component: Redis State Store

```yaml
apiVersion: dapr.io/v1alpha1
kind: Component
metadata:
  name: statestore
spec:
  type: state.redis
  metadata:
  - name: redisHost
    value: redis-master:6379
  - name: redisPassword
    value: TQJR5AQgcL
```

Components: Pub-Sub

- Hazelcast
- Redis Streams
- NATS
- Kafka
- Azure Service Bus
- RabbitMQ
- Azure Event Hubs
- GCP Pub/Sub
- MQTT

Publishing & Subscribing



POST
http://10.0.0.4:8004/order
POST
http://localhost:3500/v1.0/publish/
dapr
"email"
POST
http://10.0.0.5:8005/order
"cart"
dapr
redis
dapr
"shipping"
Publish
Subscribe

```
"topic":"order",
"data":{
  "user":"johndoe",
  "item":"ZeroDay"
},
```

```
"data":{
  "user":"johndoe",
  "item":"ZeroDay"
}
```

Component: Redis Stream Pub Sub

```yaml
apiVersion: dapr.io/v1alpha1
kind: Component
metadata:
  name: messagebus
spec:
  type: pubsub.redis
  metadata:
  - name: "redisHost"
    value: "YOUR_REDIS_HOST_HERE"
  - name: "redisPassword"
    value: "YOUR_REDIS_PASSWORD_HERE"
```

- Publish a message
  POST http://localhost:<daprPort>/v1.0/publish/<topic>

- Subscribe to to a topic. Dapr calls your app on /dapr/subscribe
  Respond with the collection of topics you want to subscribe to

- GET http://localhost:<appPort>/dapr/subscribe

  ```
  app.get('/dapr/subscribe', (_req, res) => {
      res.json([
          'A',
          'B'
      ]);
  });
  ```

- Messages are send in application/cloudevents+json format

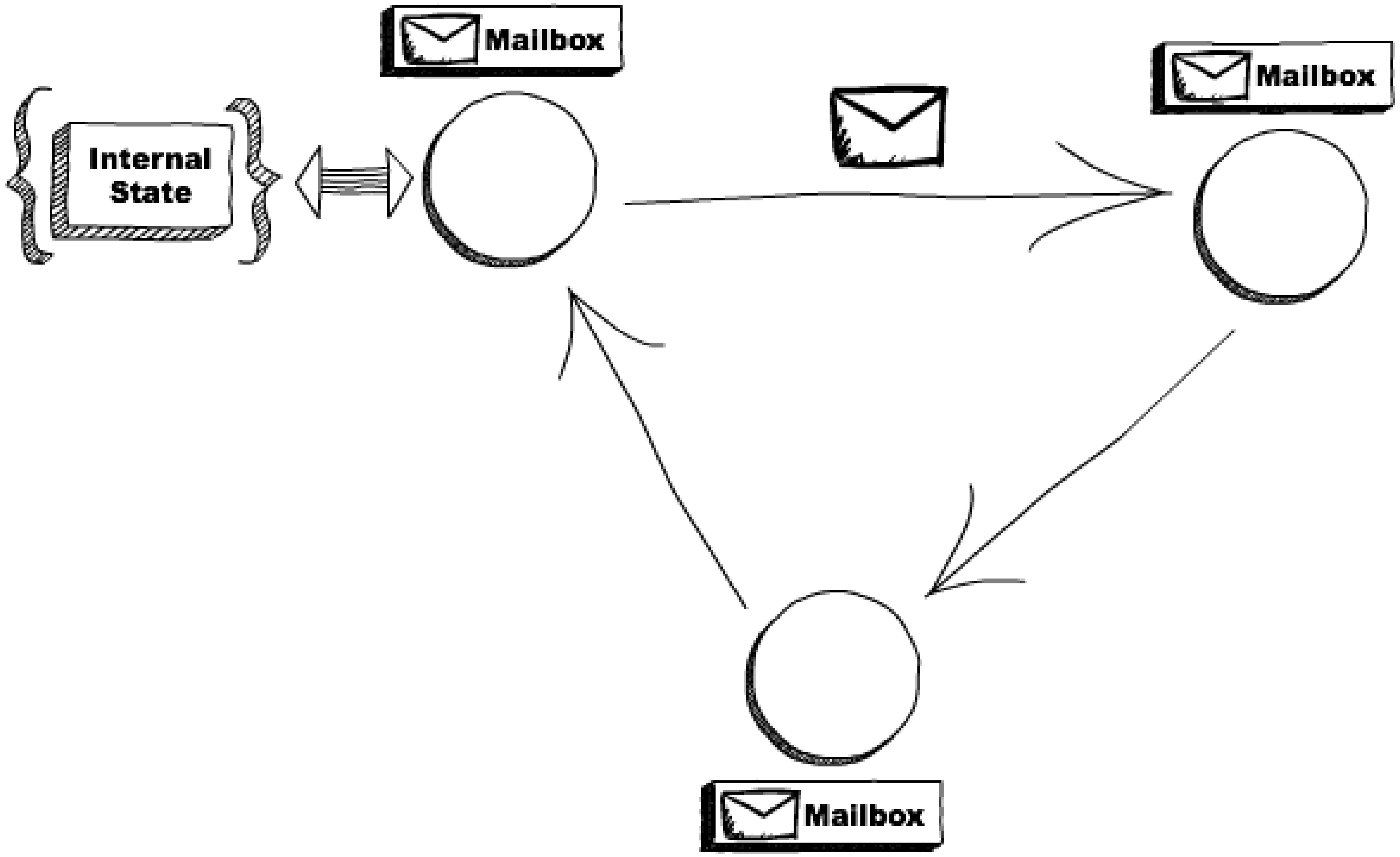
Components: Secret Store

- Kubernetes
- Hashicorp Vault
- Azure KeyVault
- AWS Secret manager
- GCP Cloud KMS
- GCP Secret Manager

Components: Tracing Exporters

- Native
  OpenTelemetry default exporter
- String
  Export to a string buffer. This is mostly used for testing purposes.
- Zipkin
  Export to a Zipkin back-end.

Actor Model

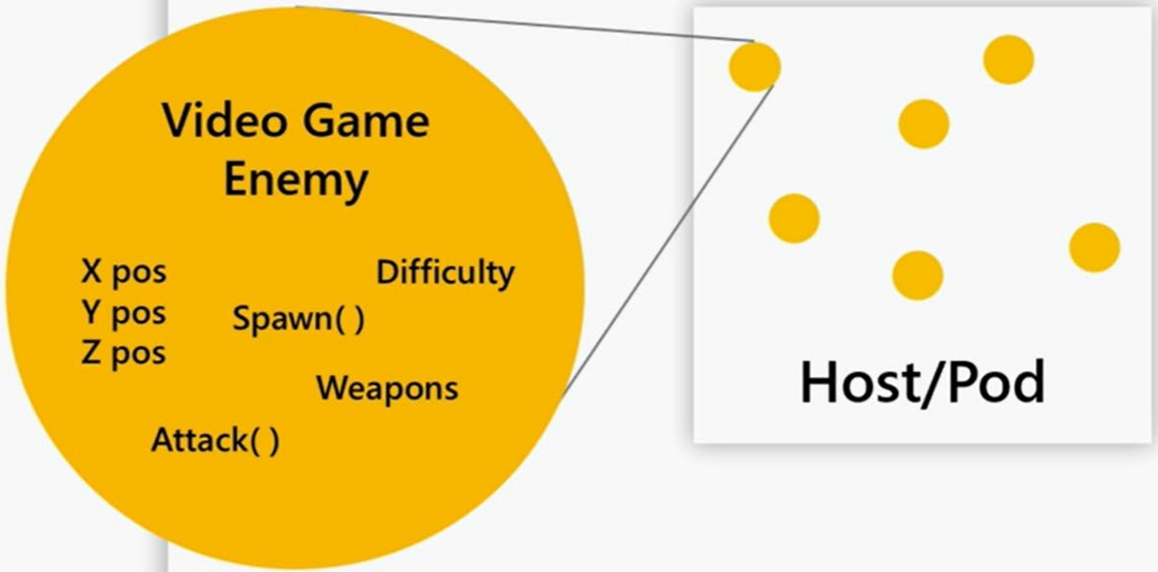


Virtual Actors with Dapr

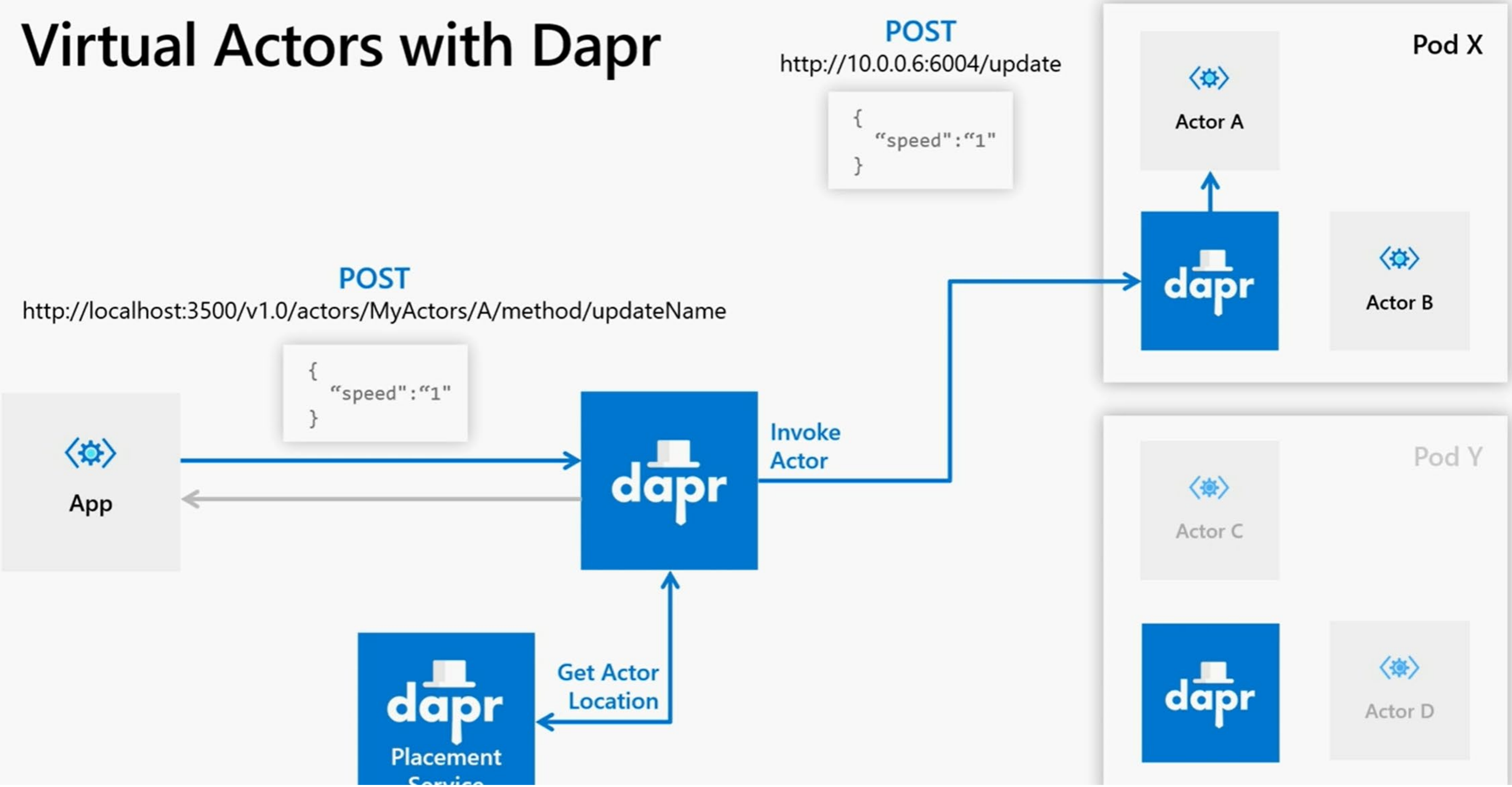
Stateful, objects of storage and compute

Dapr Actor Features:

- Distribution & failover
- Turn-based concurrency
- State management
- Timers
- Reminders



Virtual Actors with Dapr



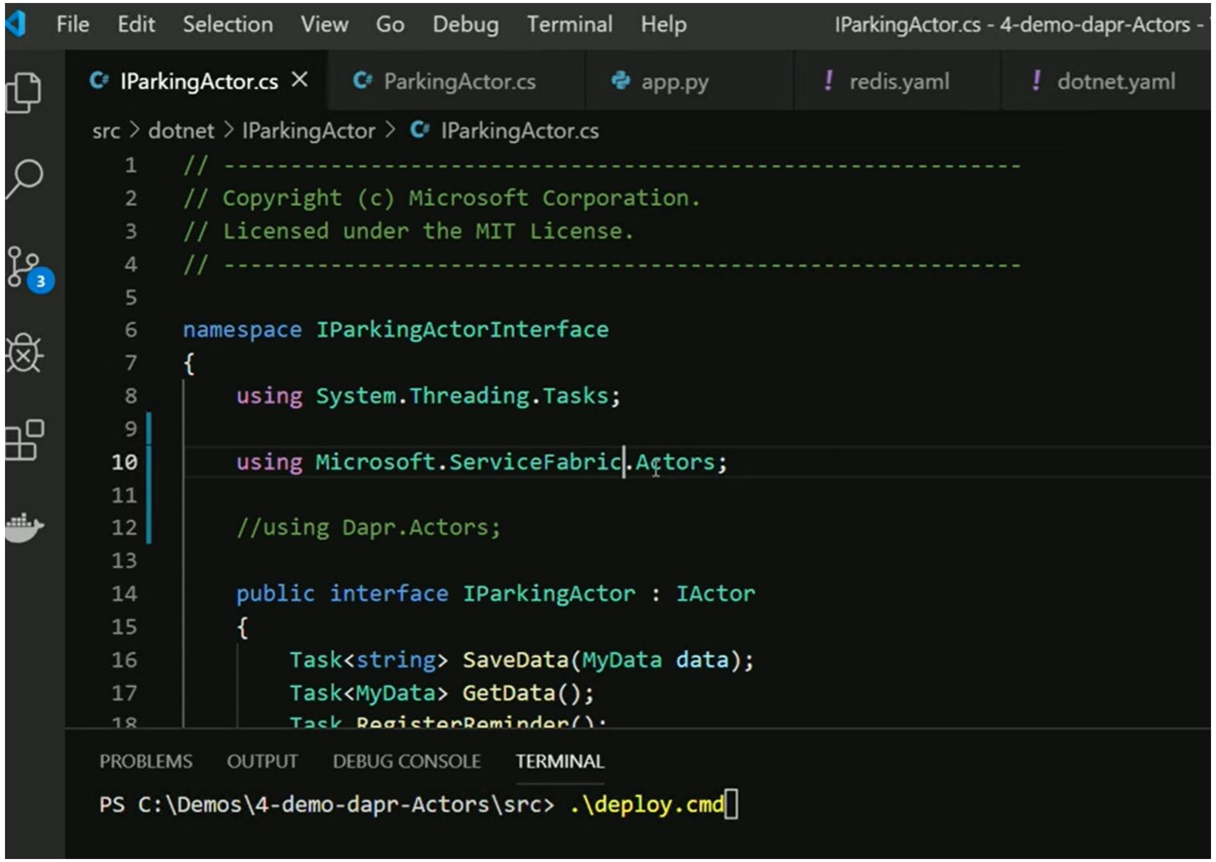
POST
http://localhost:3500/v1.0/actors/MyActors/A/method/updateName

```
{
  “speed":“1"
}
```

POST
http://10.0.0.6:6004/update

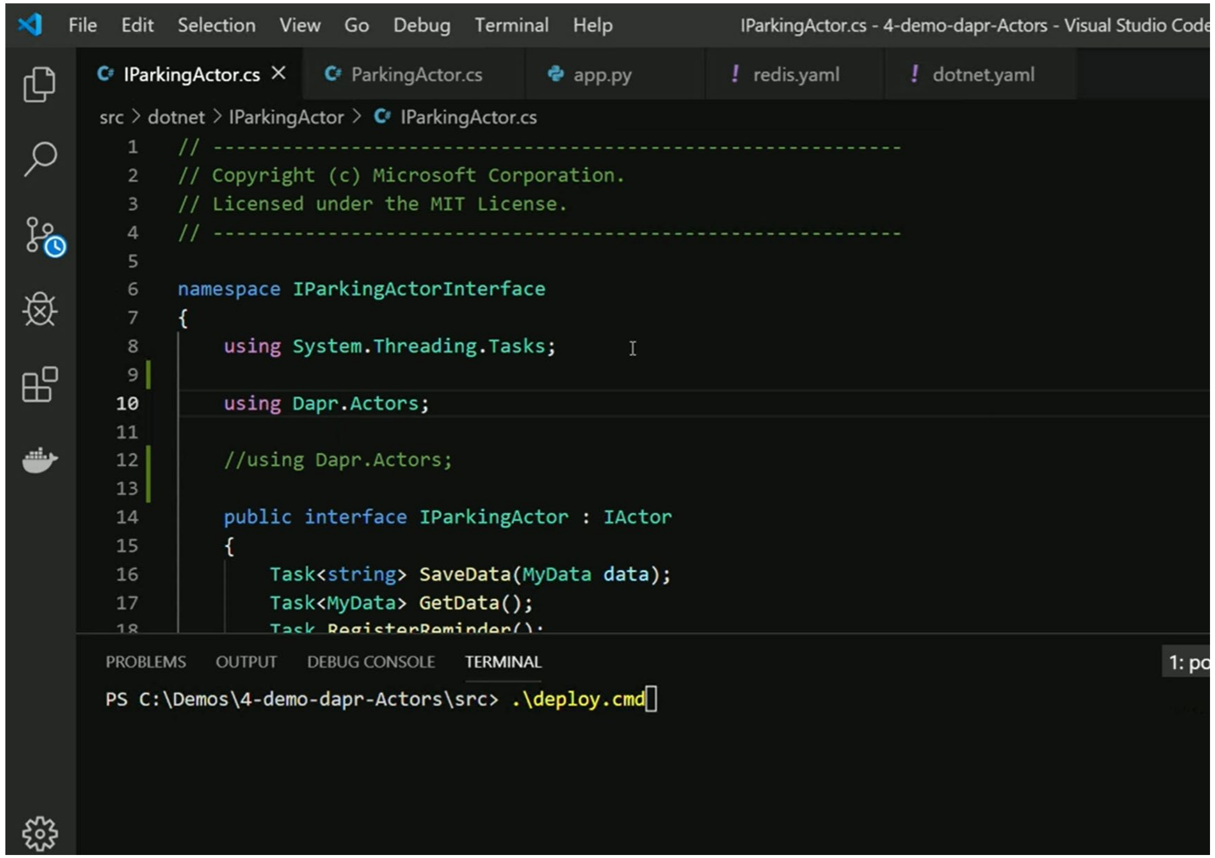
```
{
  “speed":“1"
}
```

Uses exact same ASF actor spec


File Edit Selection View Go Debug Terminal Help
IParkingActor.cs - 4-demo-dapr-Actors -
IParkingActor.cs
ParkingActor.cs
app.py
redis.yaml
dotnet.yaml
src > dotnet > IParkingActor > IParkingActor.cs
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
// ----------------------------------------------------------------
// Copyright (c) Microsoft Corporation.
// Licensed under the MIT License.
// ----------------------------------------------------------------

namespace IParkingActorInterface
{
    using System.Threading.Tasks;

    using Microsoft.ServiceFabric.Actors;

    //using Dapr.Actors;

    public interface IParkingActor : IActor
    {
        Task<string> SaveData(MyData data);
        Task<MyData> GetData();
        Task RegisterReminder();
```

```
PS C:\Demos\4-demo-dapr-Actors\src> .\deploy.cmd
```


File Edit Selection View Go Debug Terminal Help
IParkingActor.cs - 4-demo-dapr-Actors - Visual Studio Code
IParkingActor.cs
ParkingActor.cs
app.py
redis.yaml
dotnet.yaml
src > dotnet > IParkingActor > IParkingActor.cs
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
1: po

```
// ----------------------------------------------------------------
// Copyright (c) Microsoft Corporation.
// Licensed under the MIT License.
// ----------------------------------------------------------------

namespace IParkingActorInterface
{
    using System.Threading.Tasks;

    using Dapr.Actors;

    //using Dapr.Actors;

    public interface IParkingActor : IActor
    {
        Task<string> SaveData(MyData data);
        Task<MyData> GetData();
        Task RegisterReminder();
```

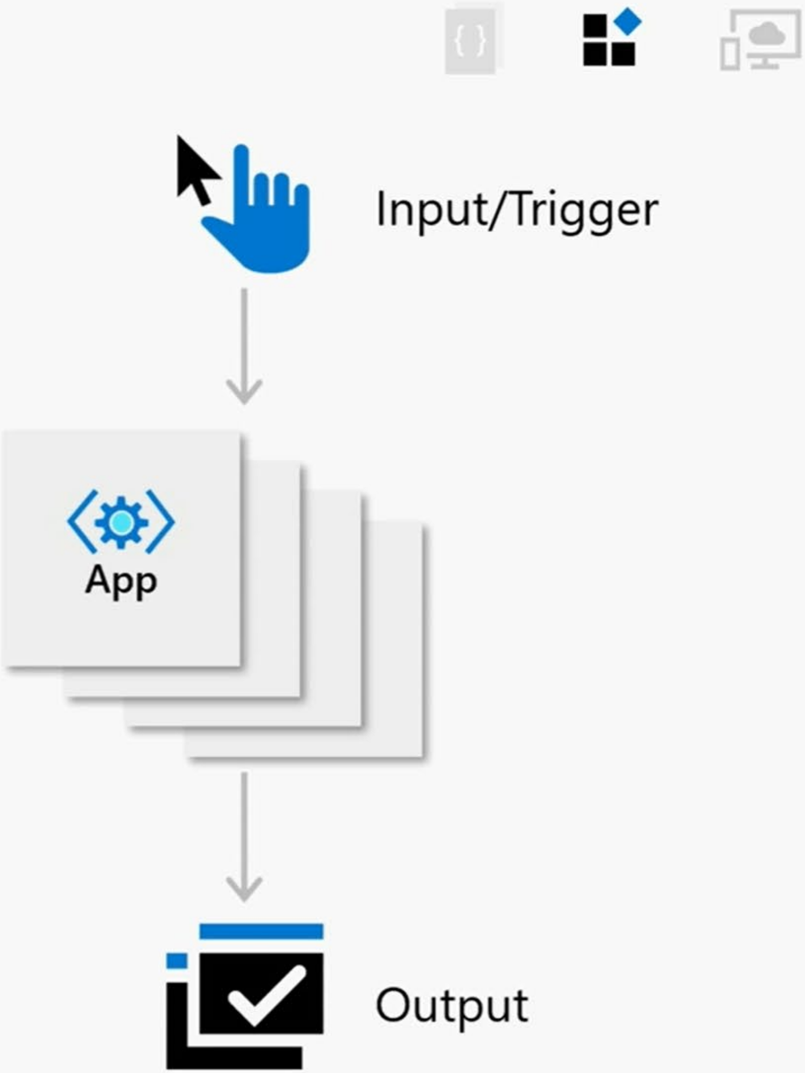
```
PS C:\Demos\4-demo-dapr-Actors\src> .\deploy.cmd
```

Functions with Dapr

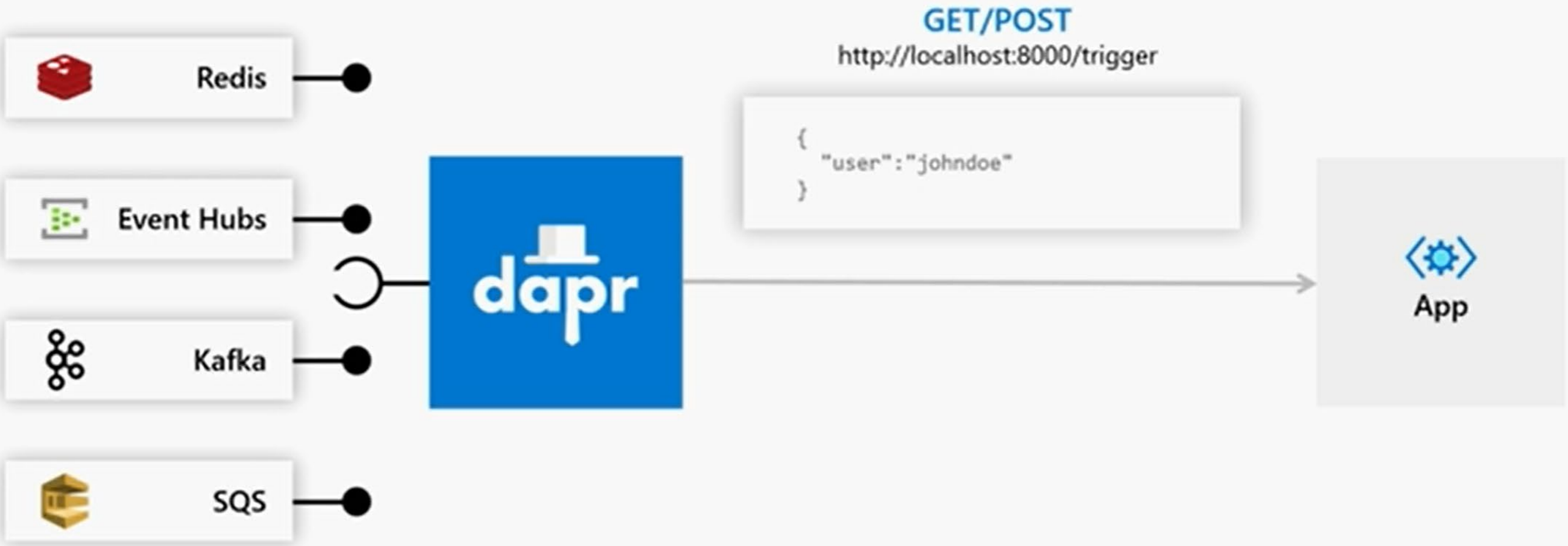
Event driven

Stateless

Easy replication/scaling



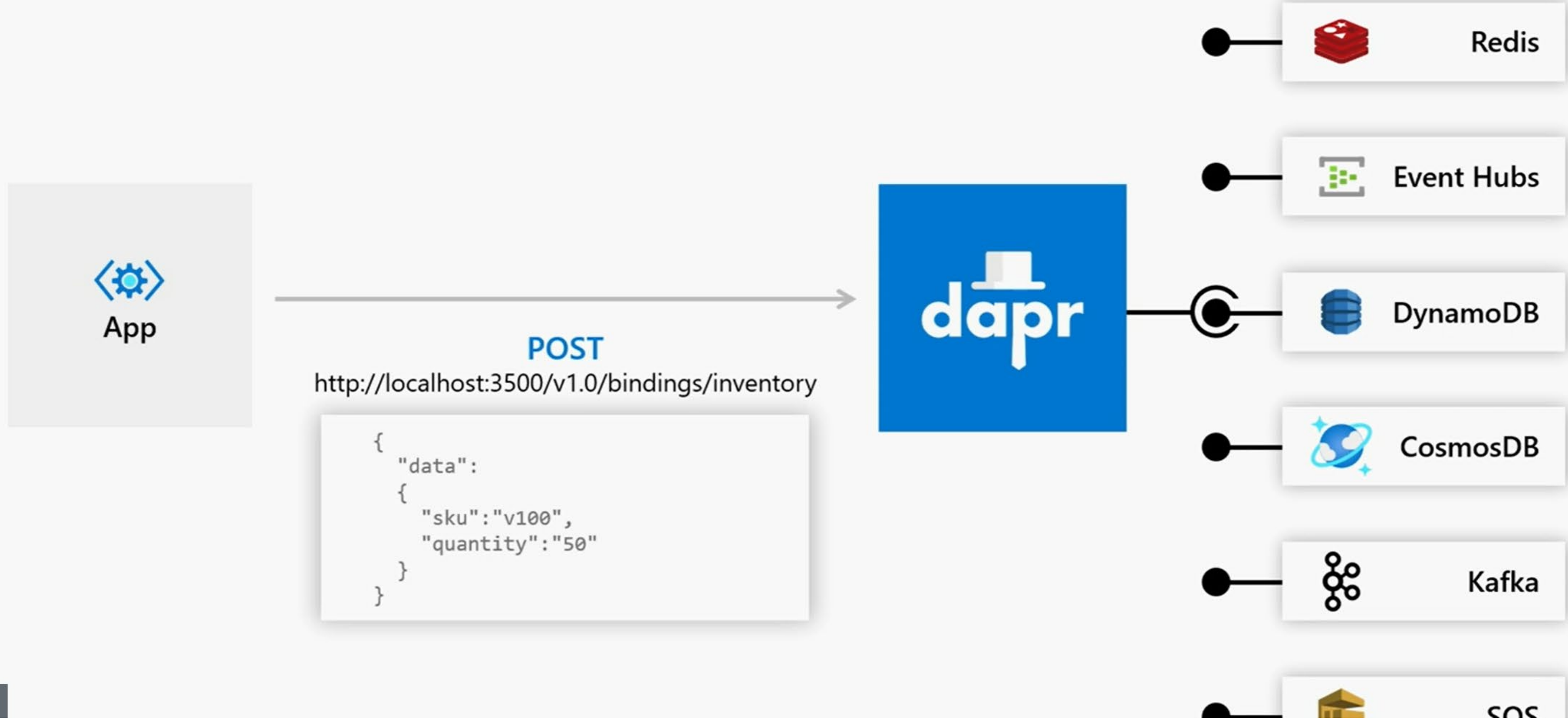
Input bindings



GET/POST
http://localhost:8000/trigger

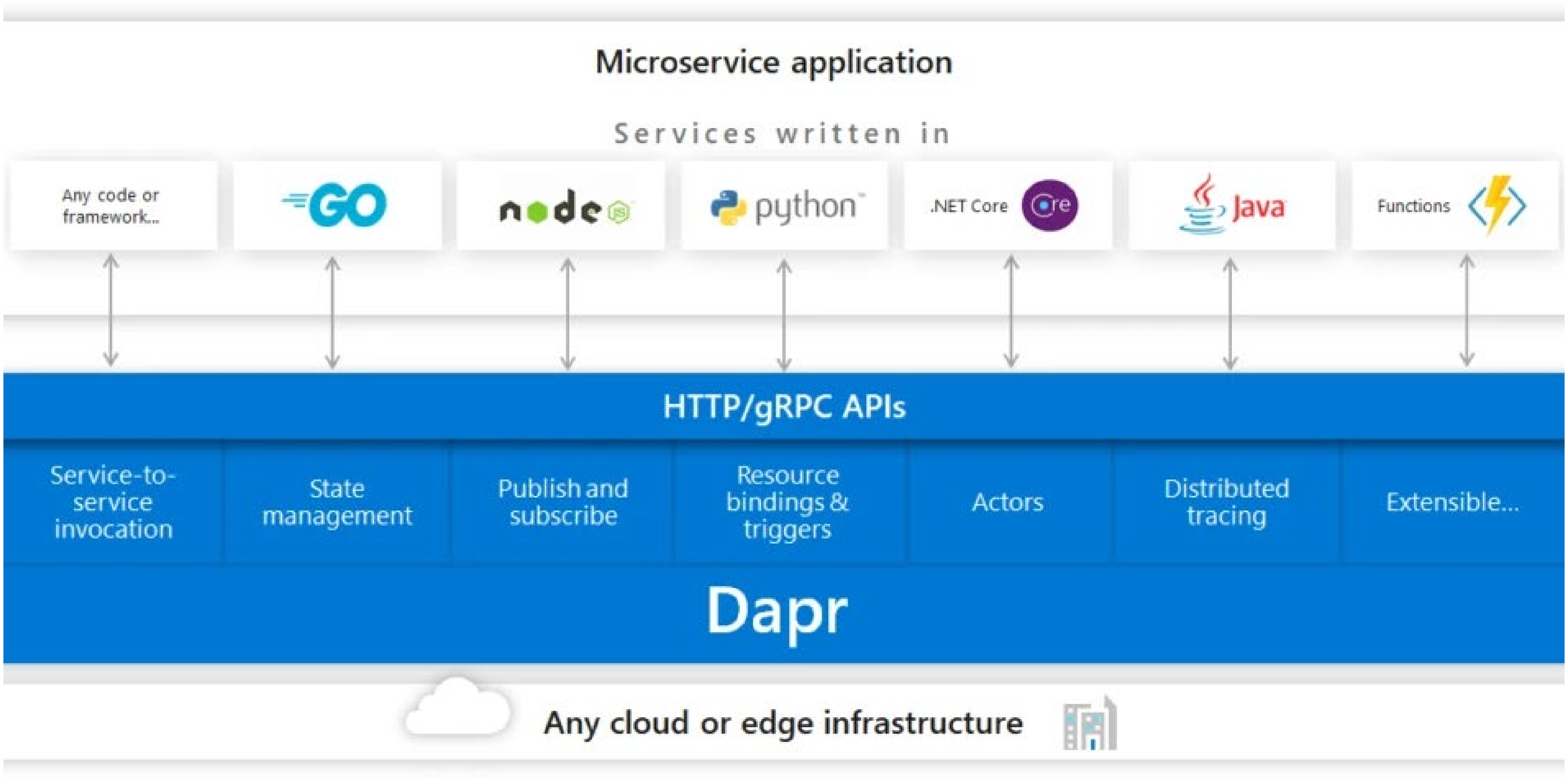
```
{
  "user":"johndoe"
}
```

Output bindings



```
{
  "data":
  {
    "sku":"v100",
    "quantity":"50"
  }
}
```

0:01:28


Microservice application
Services written in
Any code or framework...
GO
node
python
.NET Core
Java
Functions
HTTP/gRPC APIs
Service-to-service invocation
State management
Publish and subscribe
Resource bindings & triggers
Actors
Distributed tracing
Extensible...
Dapr
Any cloud or edge infrastructure

<DevSum>

Thank you!

Any questions ???

Ronald Harmsen

@ronaldharmsen

ronald@nforza.nl