

# Dapr

## Kubernetes Deep Dive

```
mirror_mod = modifier_ob.  
Set mirror object to mirror.  
mirror_mod.mirror_object =  
operation == "MIRROR_X":  
mirror_mod.use_x = True  
mirror_mod.use_y = False  
mirror_mod.use_z = False  
operation == "MIRROR_Y":  
mirror_mod.use_x = False  
mirror_mod.use_y = True  
mirror_mod.use_z = False  
operation == "MIRROR_Z":  
mirror_mod.use_x = False  
mirror_mod.use_y = False  
mirror_mod.use_z = True  
  
selection at the end -add  
mirror_ob.select= 1  
modifier_ob.select=1  
context.scene.objects.active  
("Selected" + str(modifier_ob.  
mirror_ob.select = 0  
= bpy.context.selected_object  
data.objects[one.name].select  
  
print("please select exactly  
  
-- OPERATOR CLASSES ----  
  
types.Operator):  
X mirror to the selected  
object.mirror_mirror_x"  
mirror X"  
  
context):  
context.active_object is not
```

# What's in this module?

---

What is  
Dapr?

Installing  
Dapr

Using  
Dapr



# What is Dapr?

---



dapr

The Dapr logo features a stylized blue icon above the lowercase text 'dapr'. The icon consists of a vertical rectangle with a horizontal bar extending from its top, resembling a simplified building or a server rack. The text 'dapr' is in a bold, sans-serif font.



# Distributed Application Runtime

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

dapr.io

A screenshot of the Dapr website homepage. The header includes the Dapr logo, navigation links for "Blog", "Docs", "GitHub", and "Discord", a "Star" button showing 11,791 stars, and a "Try Dapr" button. The main content area features the headline "Simplify cloud-native application development" and the subtext "Focus on your application's core logic and keep your code simple and portable". A diagram illustrates two applications, "App A" and "App B", represented as blue hexagons with code symbols "&lt; / &gt;". Both apps have a small Dapr logo next to them. They are connected by a double-headed arrow. App A is connected to a blue database cylinder below it. App B is connected to a green hexagon with a lock icon above it. A "Get Started" button is located below the subtext. A dark blue banner across the middle of the page reads "Announcing Dapr v1.0! Dapr is now production ready! Learn more &gt;&gt;". At the bottom, there is a video player titled "Introducing Dapr: The Distributed Application Runtime" with "Watch later" and "Share" buttons. The text "What is Dapr?" is visible at the bottom left of the page.

# 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

# Building blocks

HTTP API

gRPC API



Service-  
to-service  
invocation



State  
management



Publish  
and  
subscribe



Resource  
bindings  
and triggers



Actors



Observability



Secrets



Extensible

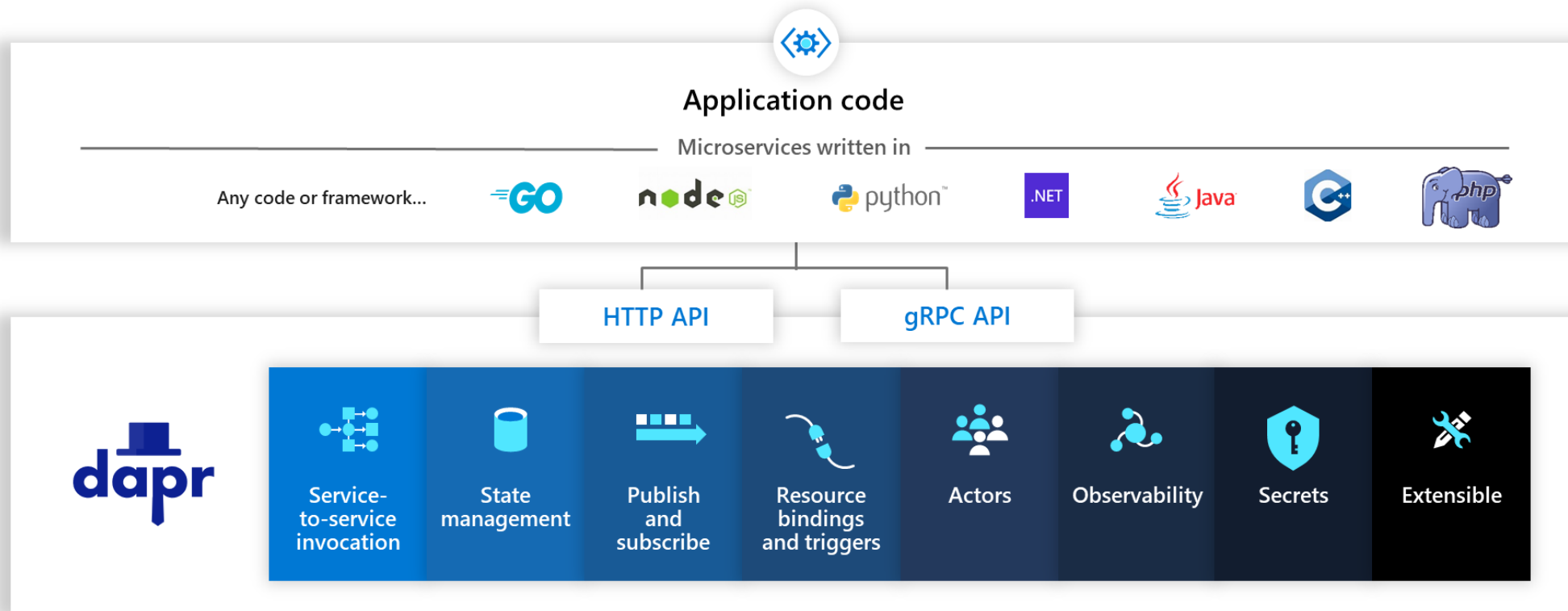
# Any language or framework

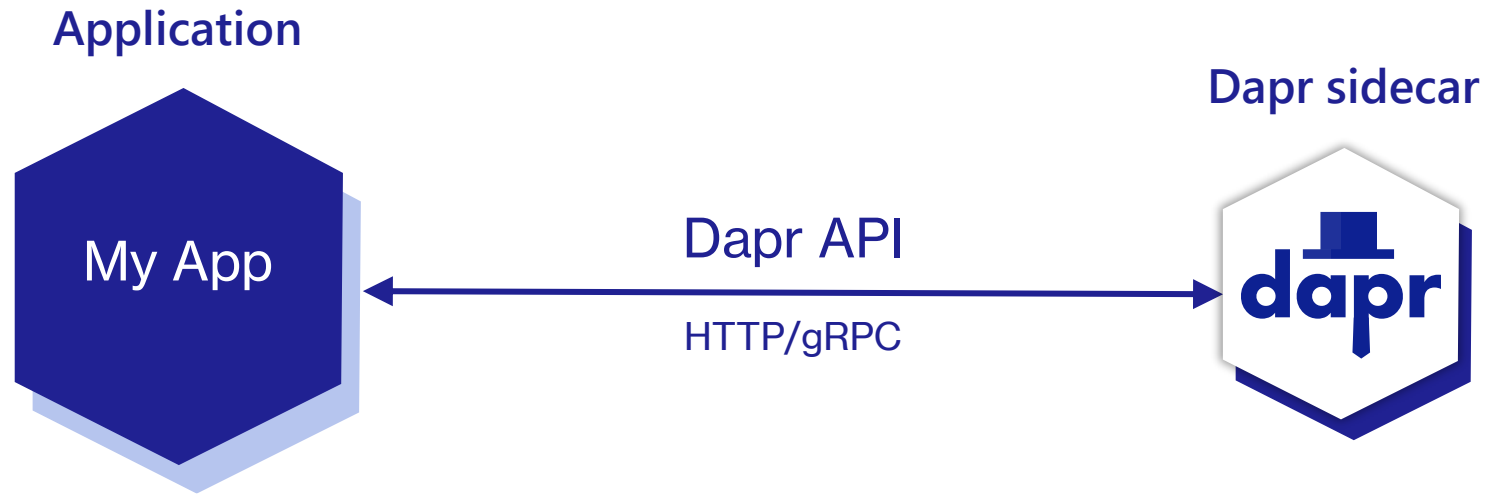


Standard APIs accessed over http/gRPC protocols from user service code



Runs as local "side car library" dynamically loaded at runtime for each service





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

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

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

**GET** `http://localhost:3500/v1.0/secrets/keyvault/password`

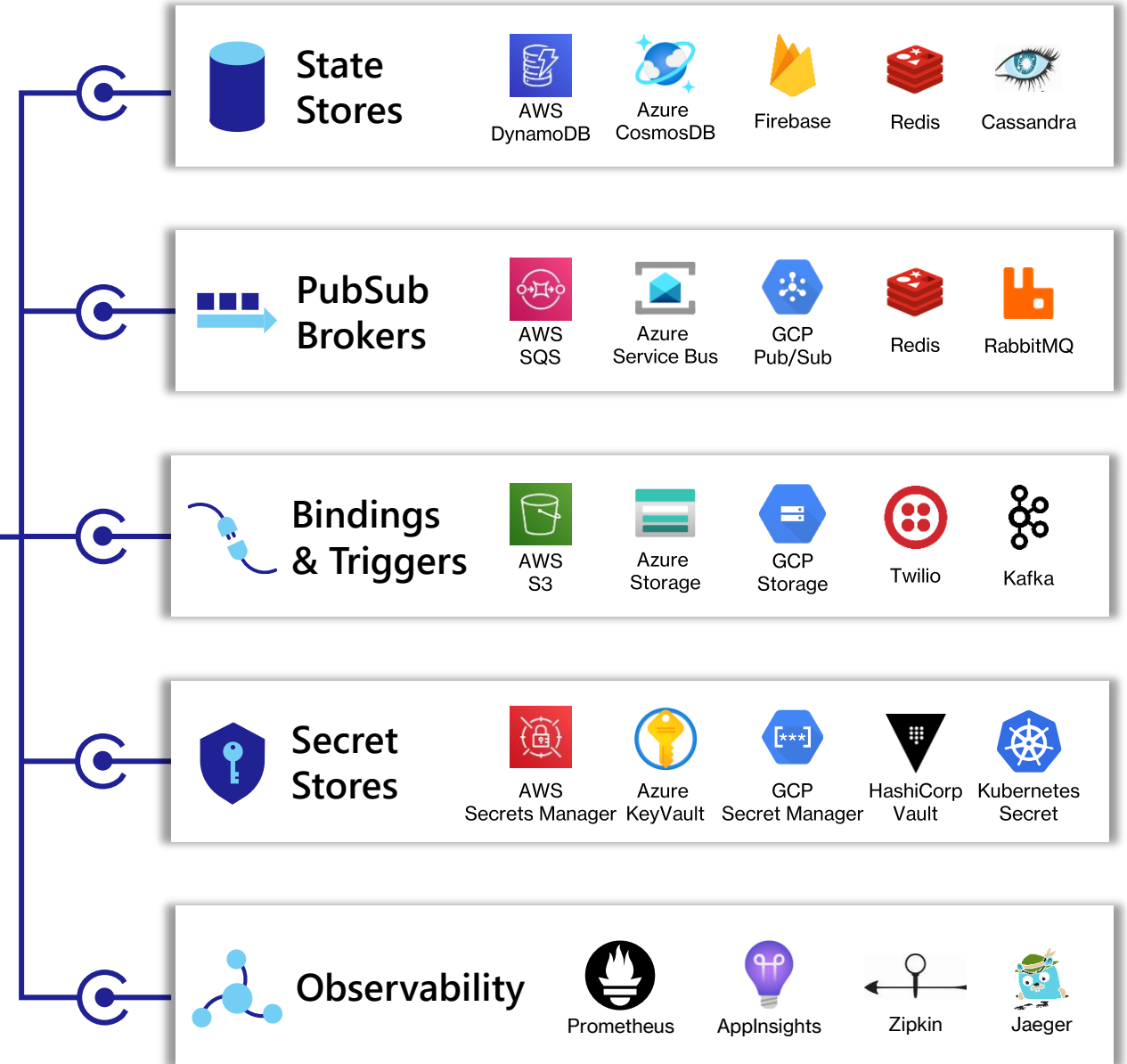




Swappable YAML files with  
resource connection details

Over 70 components available

Create components for your resource at:  
[github.com/dapr/components-contrib](https://github.com/dapr/components-contrib)





# Installing Dapr

---



# Dapr hosting environments

## Self-hosted

- Get started with `dapr init`
- Easy setup with Docker images
  - Sets up placement, Zipkin, Redis
  - `slim-init` available without Docker
- Run any application with Dapr sidecar using `dapr run`

## **kubernetes**

- Get started with `dapr init -k`
- Fully managed Dapr control plane
  - Deploys dashboard, placement, operator, sentry, and injector pods
- Automatically inject Dapr sidecar into all annotated pods
- Upgrade with `dapr upgrade` or Helm

# Install the Dapr CLI

- Runs on Linux, Windows and MacOS
- Dapr releases: <https://github.com/dapr/cli/releases>



```
wget -q https://raw.githubusercontent.com/dapr/cli/master/install/install.sh -O - | /bin/bash
```

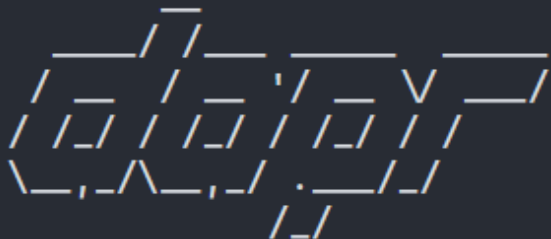
(installs to /usr/local/bin)



```
powershell -Command
```

```
"iwr -useb https://raw.githubusercontent.com/dapr/cli/master/install/install.ps1 | iex"
```

(installs to C:\dapr)



---

## Distributed Application Runtime

### Usage:

dapr [command]

### Available Commands:

build-info	Print build info of Dapr CLI and runtime
completion	Generates shell completion scripts
components	List all Dapr components. Supported platforms: Kubernetes
configurations	List all Dapr configurations. Supported platforms: Kubernetes
dashboard	Start Dapr dashboard. Supported platforms: Kubernetes and self-hosted
help	Help about any command
init	Install Dapr on supported hosting platforms. Supported platforms: Kubernetes and self-hosted
invoke	Invoke a method on a given Dapr application. Supported platforms: Self-hosted
list	List all Dapr instances. Supported platforms: Kubernetes and self-hosted
logs	Get Dapr sidecar logs for an application. Supported platforms: Kubernetes
mtls	Check if mTLS is enabled. Supported platforms: Kubernetes
publish	Publish a pub-sub event. Supported platforms: Self-hosted
run	Run Dapr and (optionally) your application side by side. Supported platforms: Self-hosted
status	Show the health status of Dapr services. Supported platforms: Kubernetes
stop	Stop Dapr instances and their associated apps. Supported platforms: Self-hosted
uninstall	Uninstall Dapr runtime. Supported platforms: Kubernetes and self-hosted
upgrade	Upgrades or downgrades a Dapr control plane installation in a cluster. Supported platforms: Kubernetes

# Initialize Dapr

- Run **dapr init**
  - Fetches sidecar binaries
  - Runs a few containers: redis, zipkin, placement
  - Creates a default components folder
- Verify Dapr version  
**dapr --version**
- Verify containers  
**docker container ls**

```
gbaeke ~ .dapr $ tree -L 2
.
├── bin
│   ├── daprd
│   ├── dashboard
│   └── web
├── components
│   ├── pubsub.yaml
│   └── statestore.yaml
└── config.yaml

3 directories, 5 files
```

```
apiVersion: dapr.io/v1alpha1
kind: Component
metadata:
  name: pubsub
spec:
  type: pubsub.redis
  version: v1
  metadata:
    - name: redisHost
      value: localhost:6379
    - name: redisPassword
      value: ""
```

# dapr Dashboard

## Configuration: daprConfig

Summary



Configuration

### Summary

Name	daprConfig
Kind	Configuration
Created	2021-10-20 17:23.22
Age	20h
Tracing Enabled	true
Sampling Rate	1
Metrics Enabled	true
MTLS Enabled	false

# dapr Dashboard

## Dapr Components

	Name	Type	Age	Created
	pubsub	pubsub.redis	20h	2021-10-20 17:23.22
	statestore	state.redis	20h	2021-10-20 17:23.22

# dapr Dashboard

Scope  
All

## Overview

### Dapr Applications

Name	Age	Actions
myapp	6m	<button>Stop</button>

# Follow along

---

- Installing dapr
- Initialize dapr
- Running the sidecar
- Saving state in Redis



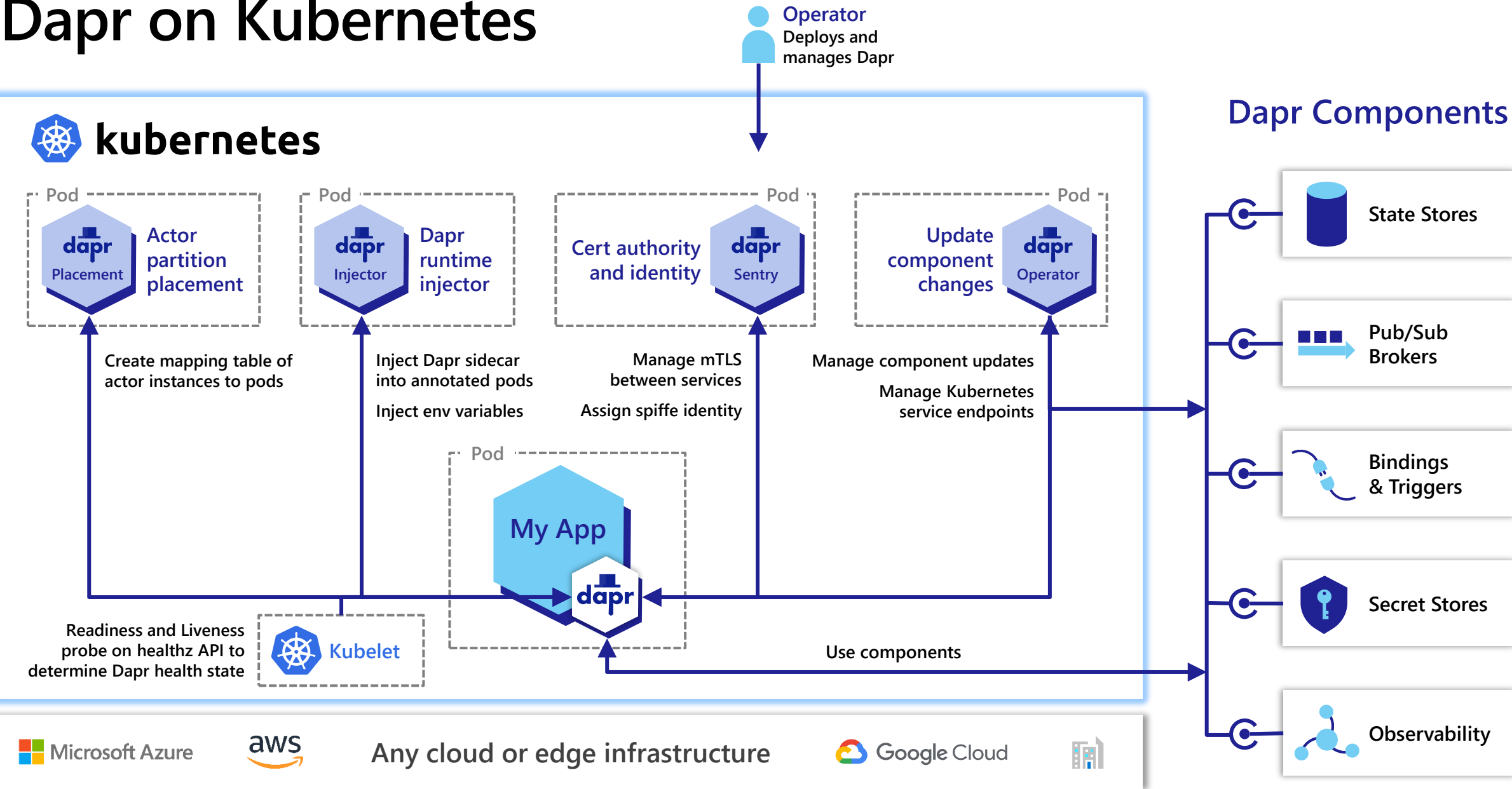


# Install on Kubernetes

- On a development cluster: **dapr init --kubernetes --wait**
- In production, use Helm:

```
helm upgrade --install dapr dapr/dapr \
--version=1.4 \
--namespace dapr-system \
--create-namespace \
--set global.ha.enabled=true \
--wait
```

# Dapr on Kubernetes



# No default components on K8S

- You need to decide which components to use and create them with YAML
- For example: Redis
  - Install Redis in Kubernetes – or – use Azure Redis Cache
  - Create a secret to hold the Redis password
  - Create a state store component for Redis

```
apiVersion: dapr.io/v1alpha1
kind: Component
metadata:
  name: statestore
  namespace: default
spec:
  type: state.redis
  version: v1
  metadata:
    - name: redisHost
      value: redis-master.default.svc.cluster.local:6379
    - name: redisPassword
      secretKeyRef:
        name: redis
        key: redis-password
```

# Follow along

---

- Installing Dapr on K8S
- Check what was deployed
- View the dashboard





# Using Dapr

---



# Coming up...



Service invocation



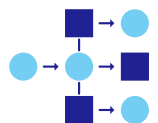
State Management



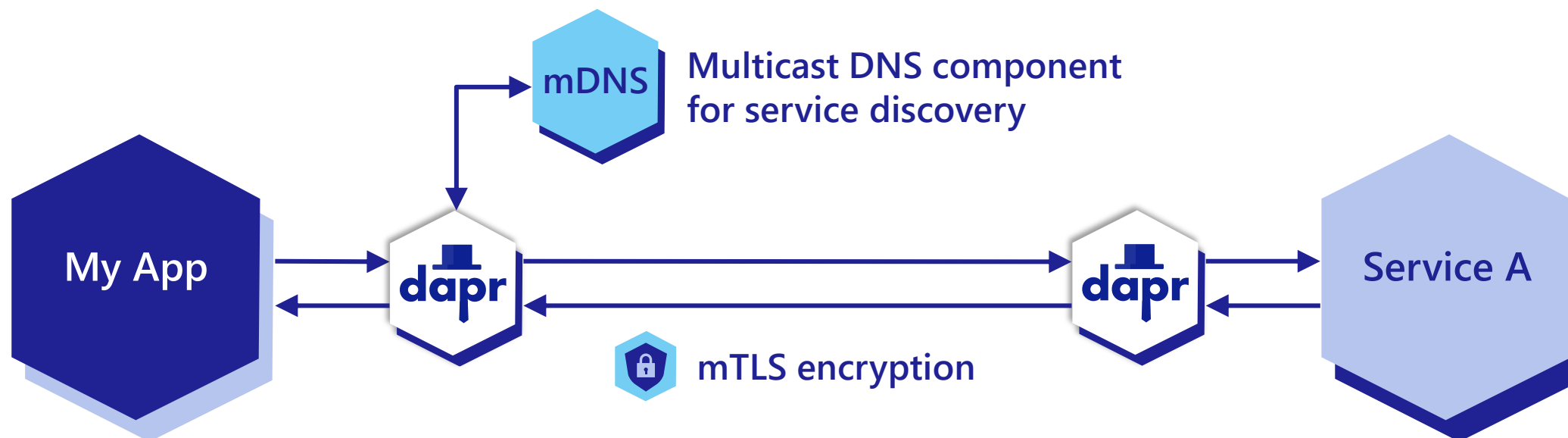
Pub/sub



Bindings



# Service invocation



POST

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

```
{"data": "Hello World"}
```

POST

`http://10.0.0.2:8000/neworder`

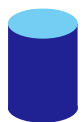
```
{"data": "Hello World"}
```

```
public async Task<VehicleInfo> GetVehicleInfo(string licenseNumber)
{
    return await _httpClient.GetFromJsonAsync<VehicleInfo>(
        $"http://localhost:6002/vehicleinfo/{licenseNumber}");
}
```

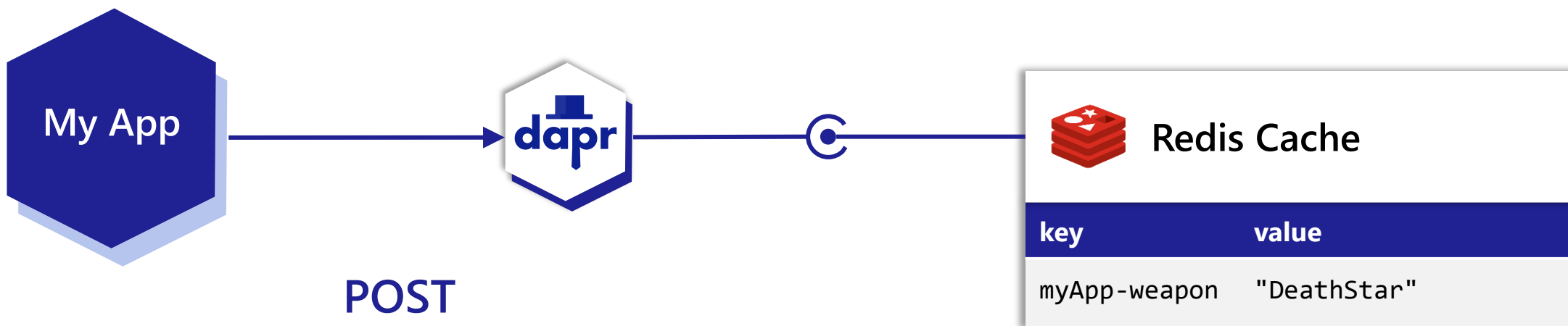


```
public async Task<VehicleInfo> GetVehicleInfo(string licenseNumber)
{
    return await _httpClient.GetFromJsonAsync<VehicleInfo>(
        $"http://localhost:3601/v1.0/invoke/vehicleregistrationservice/method/vehicleinfo/{licenseNumber}");
}
```





# State management

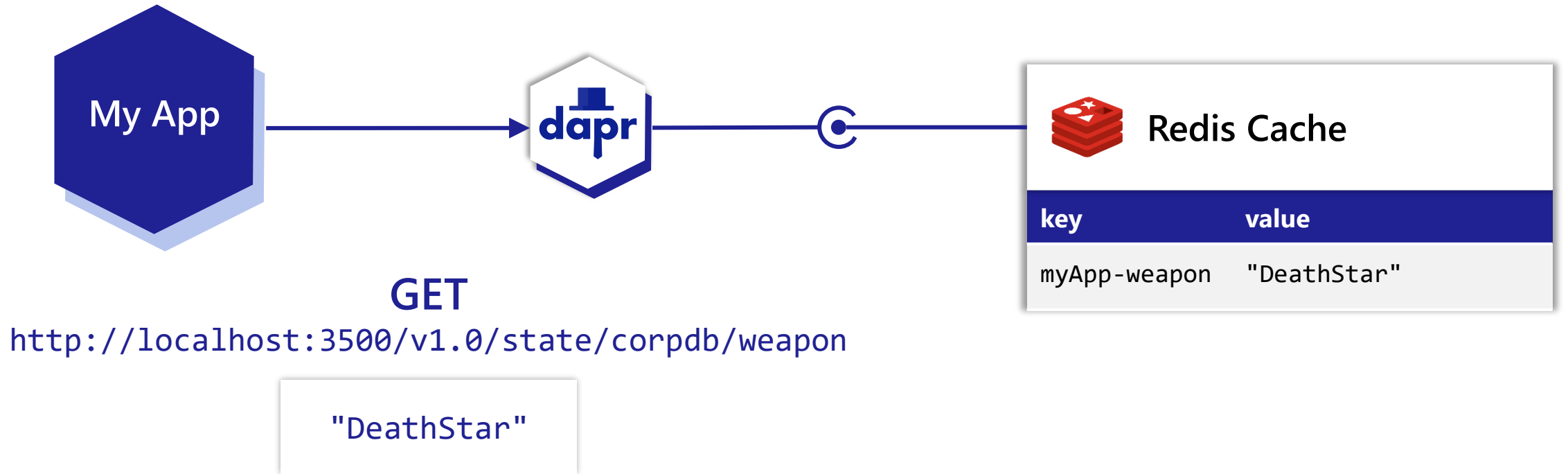


**POST**  
`http://localhost:3500/v1.0/state/corpdb`

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

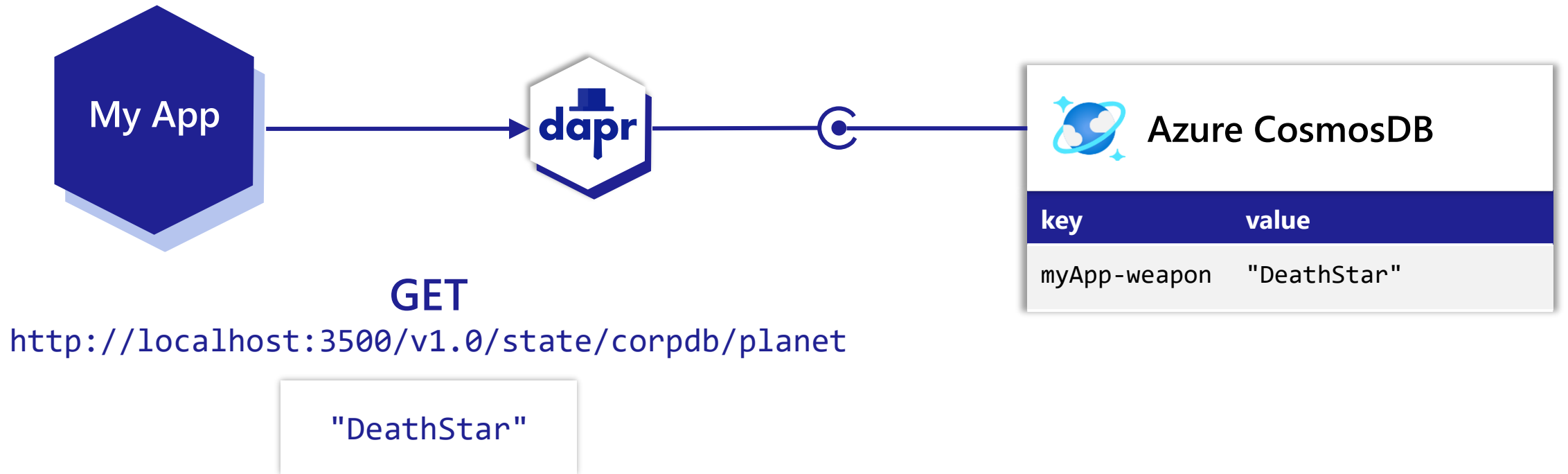


# State management





# State management



# Dapr state API

## Save state

POST /v1.0/state/corpdb

## Retrieve state

GET /v1.0/state/corpdb/mystate

## Delete state

DELETE /v1.0/state/corpdb/mystate

## Get bulk state

POST /v1.0/state/corpdb/bulk

## Submit multiple state transactions

POST /v1.0/state/corpdb/transaction

### corpdb-redis.yaml

```
apiVersion: dapr.io/v1alpha1
kind: Component
metadata:
  name: corpdb
spec:
  type: state.redis
  version: v1
  metadata:
    - name: redisHost
      value: redis-master.default.svc.cluster.local:6379
    - name: redisPassword
      secretKeyRef:
        name: redis-secret
        key: redis-password
```

# Dapr state API

## Save state

POST /v1.0/state/corpdb

## Retrieve state

GET /v1.0/state/corpdb/mystate

## Delete state

DELETE /v1.0/state/corpdb/mystate

## Get bulk state

POST /v1.0/state/corpdb/bulk

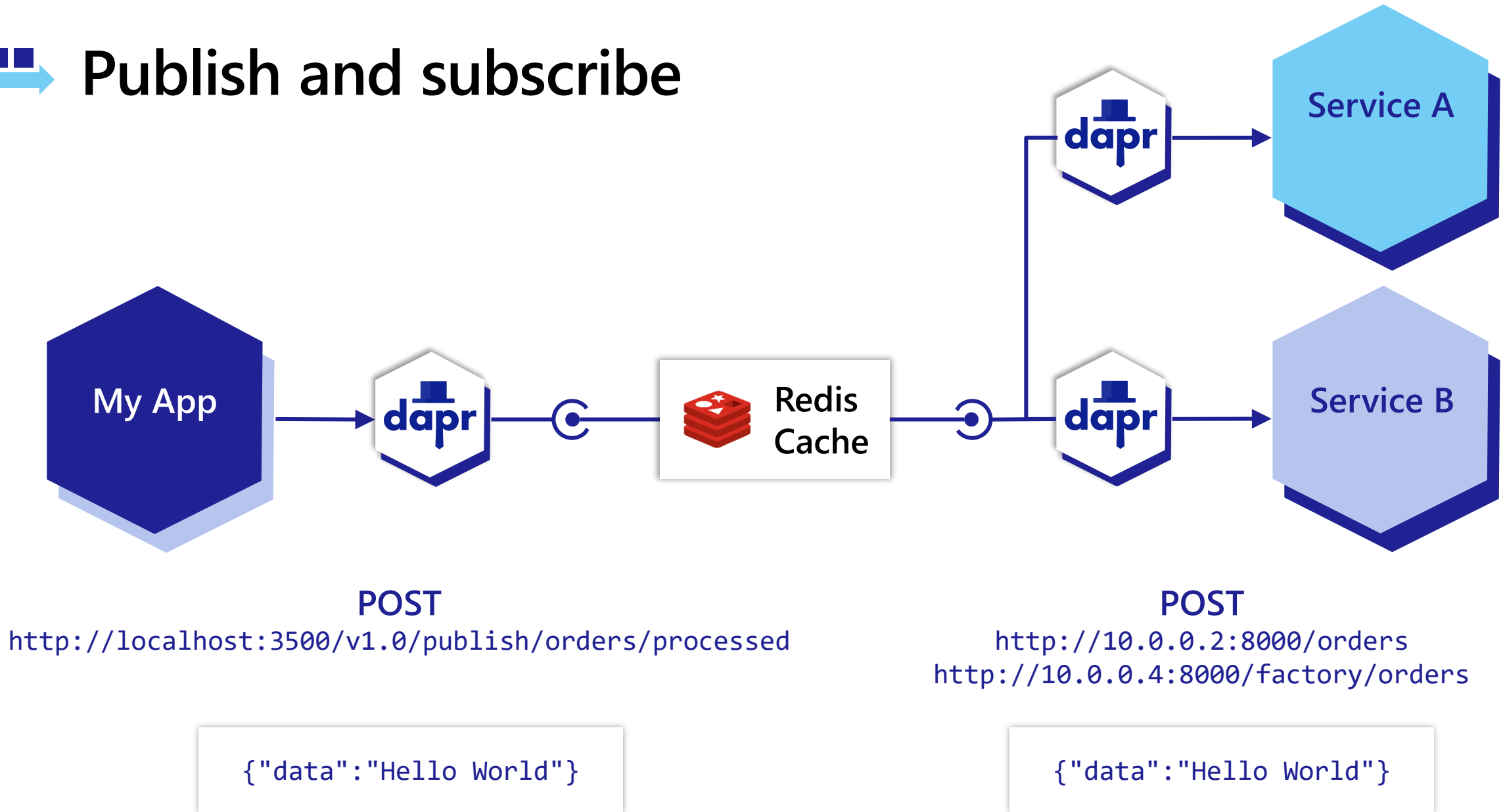
## Submit multiple state transactions

POST /v1.0/state/corpdb/transaction

### corpdb-cosmosdb.yaml

```
apiVersion: dapr.io/v1alpha1
kind: Component
metadata:
  name: corpdb
spec:
  type: state.azure.cosmosdb
  version: v1
  metadata:
    - name: url
      value: corpdb.documents.azure.com
    - name: masterKey
      secretKeyRef:
        name: master-key
        key: cosmos-key
    - name: database
      value: orders
    - name: collection
      value: processed
```

## Publish and subscribe



# Dapr pub/sub API

App-to-sidecar

## Publish a message

POST /v1.0/publish/orders/processed

Sidecar-to-app

## Get app subscriptions

GET /dapr/subscribe

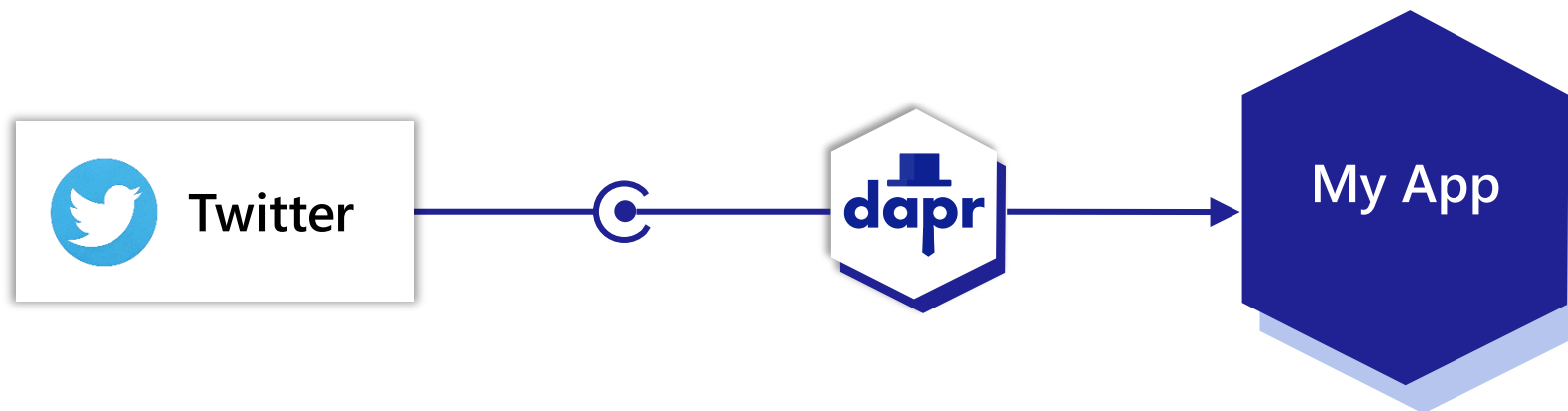
## Publish to app

POST /order-processing

orders.yaml

```
apiVersion: dapr.io/v1alpha1
kind: Component
metadata:
  name: orders
spec:
  type: pubsub.redis
  metadata:
    - name: redisHost
      value: leader.redis.svc.cluster.local:6379
    - name: redisPassword
      secretKeyRef:
        name: redis-secret
        key: password
    - name: allowedTopics
      value: "processed,audit"
```

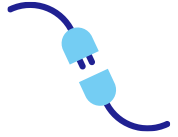
# Input triggers



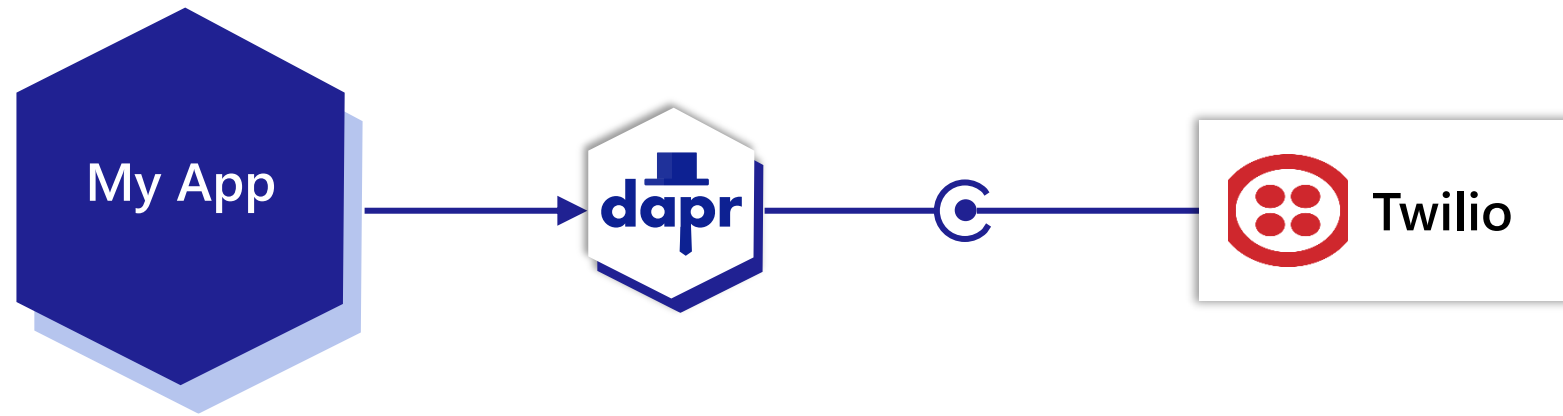
**POST**  
`http://10.0.0.2:8000/newtweet`

```
{"data": "🔊 We are excited  
to announce the ..."} 
```





# Output bindings



**POST**

`http://localhost:3500/v1.0/bindings/twilio`

```
{"data": "Hello World"}
```

Hello World

# Dapr bindings API

App-to-sidecar

## Invoke an output binding

POST/PUT /v1.0/bindings/twitter

Sidecar-to-app

## Trigger an app

OPTIONS/POST /new-tweet

```
apiVersion: dapr.io/v1alpha1
kind: Component
metadata:
  name: twitter
spec:
  type: bindings.twitter
  version: v1
  metadata:
    - name: consumerKey
      secretKeyRef:
        name: twitter-secret
        key: consumerKeys
    - name: consumerSecret
      secretKeyRef:
        name: twitter-secret
        key: consumerSecret
    - name: accessToken
      secretKeyRef:
        name: twitter-secret
        key: accessToken
    - name: accessSecret
      secretKeyRef:
        name: twitter-secret
        key: accessSecret
```

# Let's try it!

---

- Pub sub with Node.js
- Using an output binding (Azure storage)
- Secrets with Key Vault and AAD Pod Identity (optional)

