Dapr

Distributed Application Runtime

Philip Chen
Sr. Corp Cloud Solution Architect
Microsoft, Customer Success, Hybrid & Edge Strategy

Modern microservice architectures

好擴充
Deploying scale-out apps for flexibility, cost, and efficiency

服務互動
Developing resilient, scalable, microservice-based apps that interact with services

專注在應用開發
Focusing on building applications, not infrastructure

程式簡化
Trending toward serverless platforms with simple code to cloud pipelines

多程式語言
Using multiple languages and frameworks during development

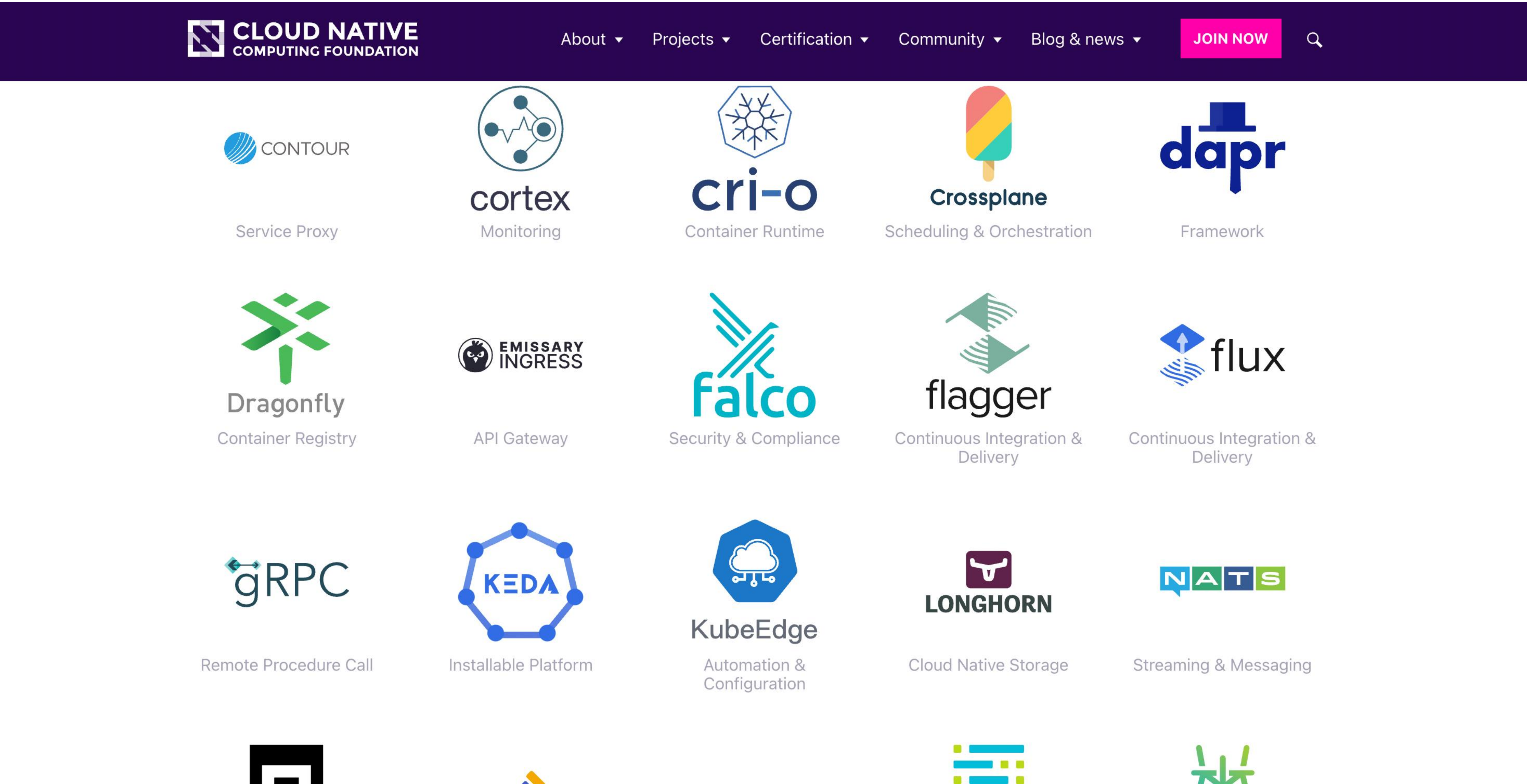
Common challenges in microservice development

複雜的開發工具
Disjointed tools and runtimes to build distributed applications

執行環境耦合過高
Runtimes have limited language support and tightly controlled feature sets

無法跨雲整合
Runtimes only target specific infrastructure platforms with limited portability

cncf.io/projects/dapr



CLOUD NATIVE COMPUTING FOUNDATION

About ▾ Projects ▾ Certification ▾ Community ▾ Blog & news ▾ JOIN NOW

CONTOUR
Service Proxy

cortex
Monitoring

cri-o
Container Runtime

Crossplane
Scheduling & Orchestration

dapr
Framework

Dragonfly
Container Registry

EMISSARY INGRESS
API Gateway

falco
Security & Compliance

flagger
Continuous Integration & Delivery

flux
Continuous Integration & Delivery

gRPC
Remote Procedure Call

KEDA
Installable Platform

KubeEdge
Automation & Configuration

LONGHORN
Cloud Native Storage

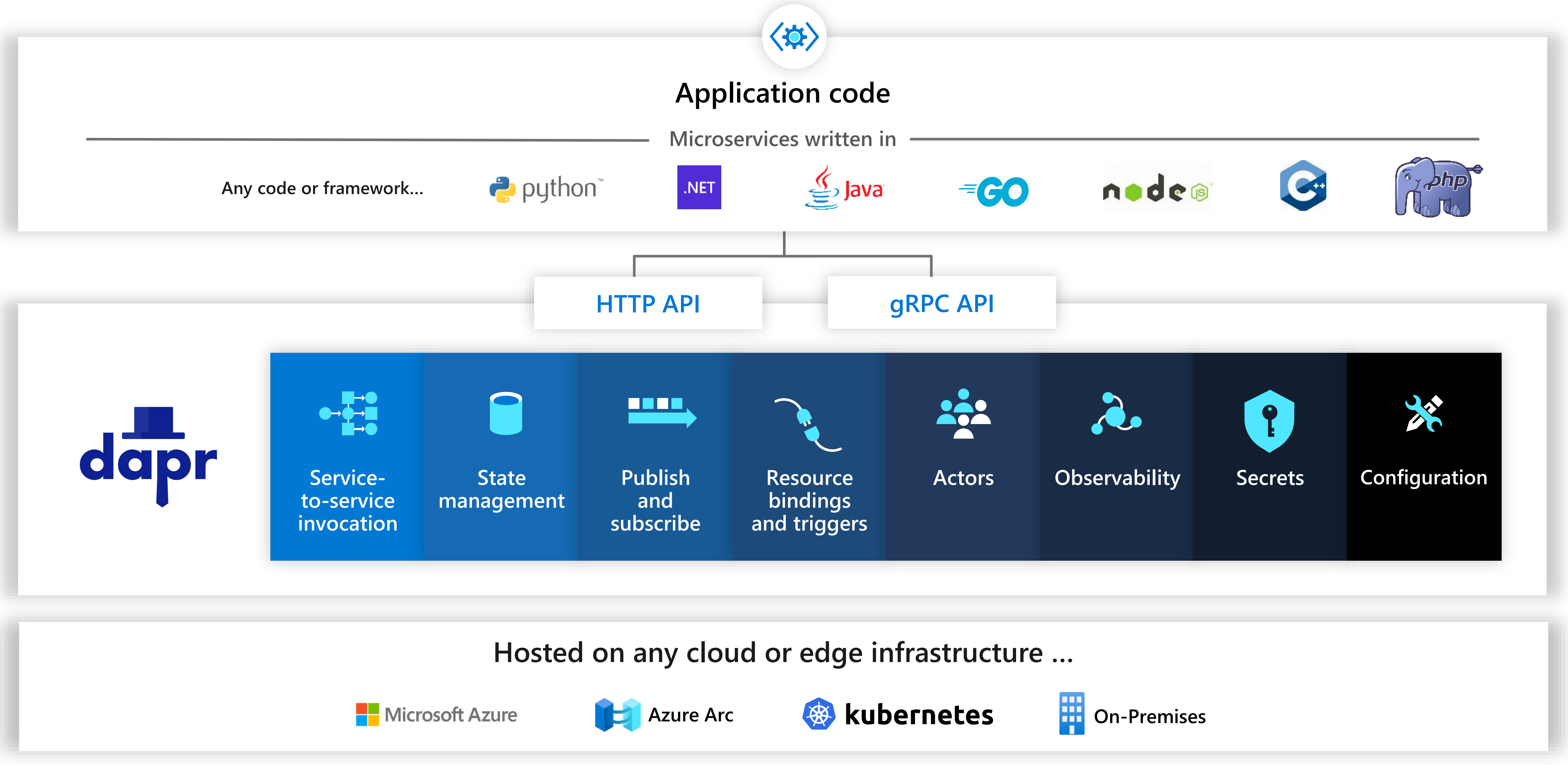
NATS
Streaming & Messaging

Dapr value pillars

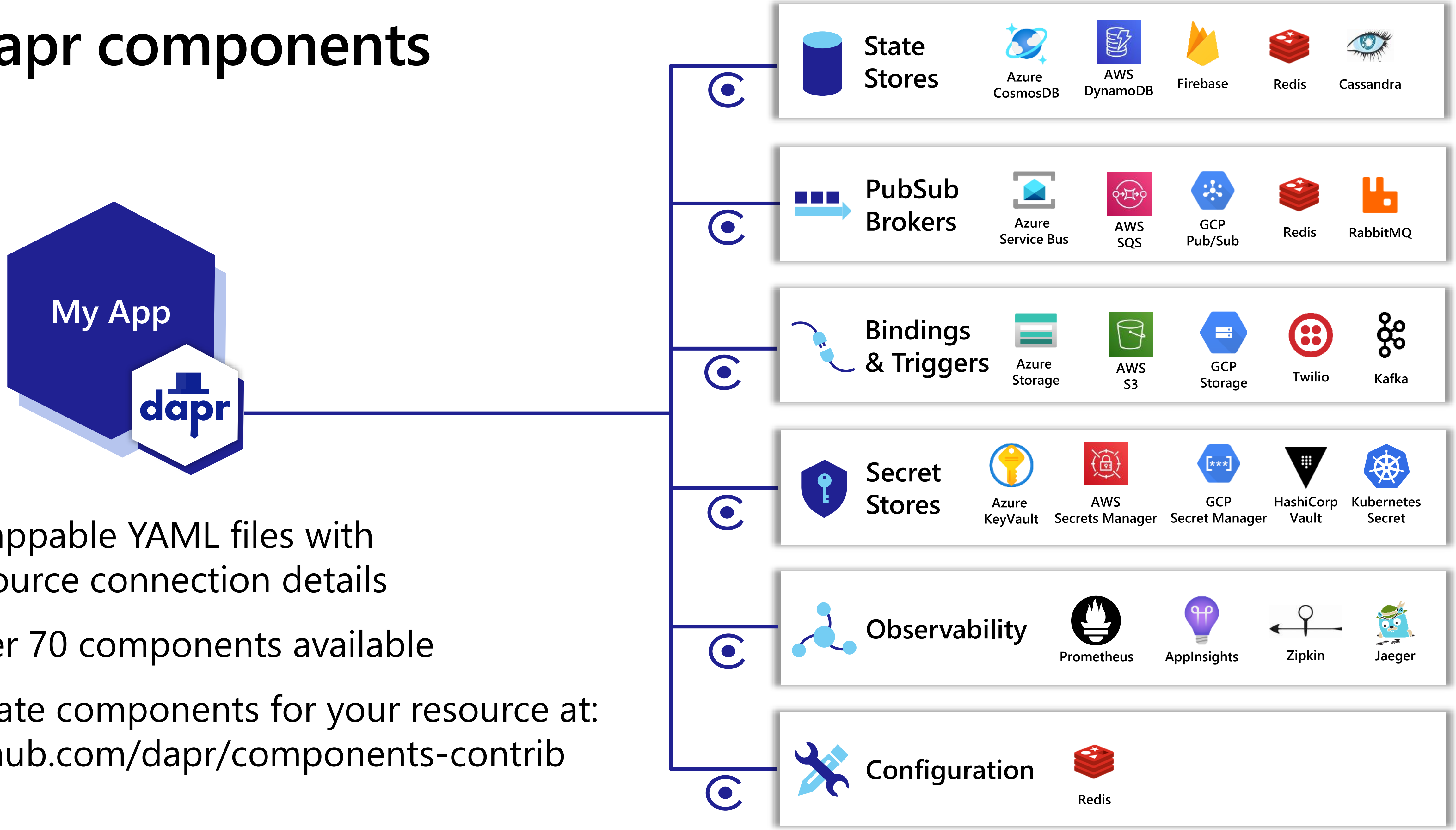
- Best-practices building blocks
- Consistent, portable, open APIs
- Extensible and pluggable components
- Any language or framework
- Adopt standards
- Platform agnostic cloud + edge

- Community driven, vendor neutral

Microservice building blocks



Dapr components

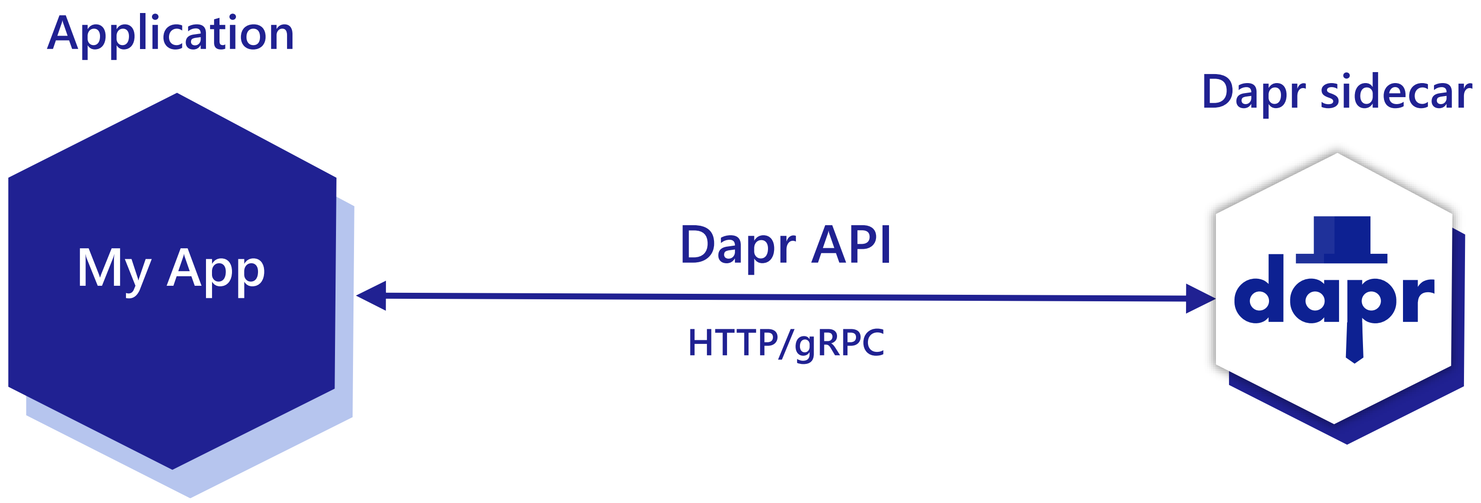


Swappable YAML files with resource connection details

Over 70 components available

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

Sidecar model 邊車模型



```
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
```

Dapr Sidecar yaml sample

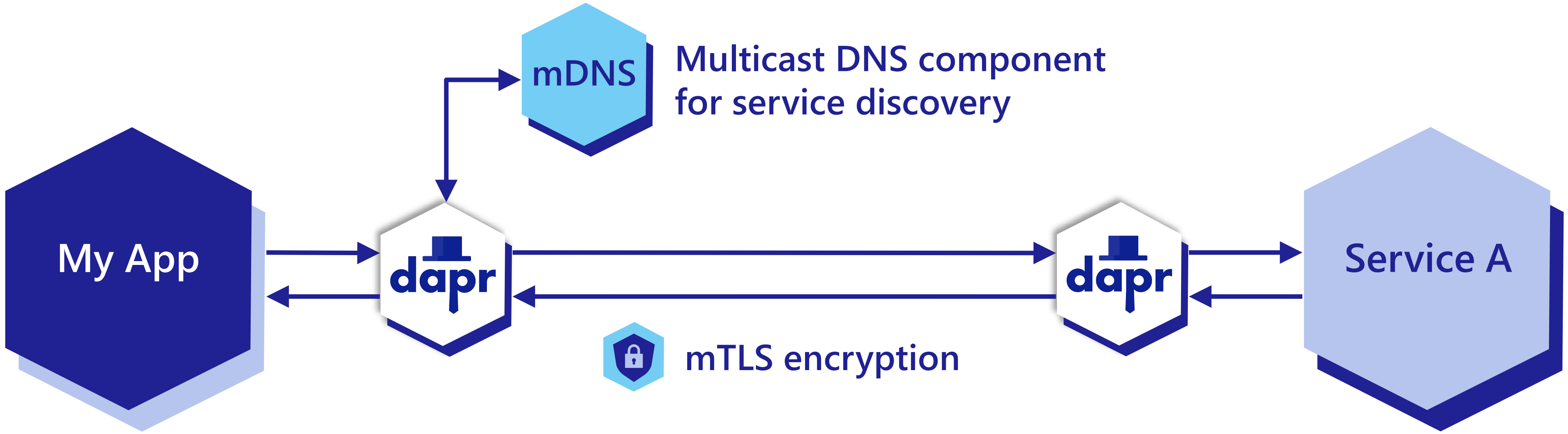
dapr.io/enabled: true - this tells the Dapr control plane to inject a sidecar to this deployment

dapr.io/app-id: pythonapp - this assigns a unique ID or name to the Dapr application, so it can be sent messages to and communicated with by other Dapr apps

python.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: pythonapp
  labels:
    app: python
spec:
  replicas: 1
  selector:
    matchLabels:
      app: python
  template:
    metadata:
      labels:
        app: python
      annotations:
        dapr.io/enabled: "true"
        dapr.io/app-id: "pythonapp"
        dapr.io/enable-api-logging: "true"
    spec:
      containers:
      - name: python
        image: ghcr.io/dapr/samples/hello-k8s-python:latest
```

Service invocation



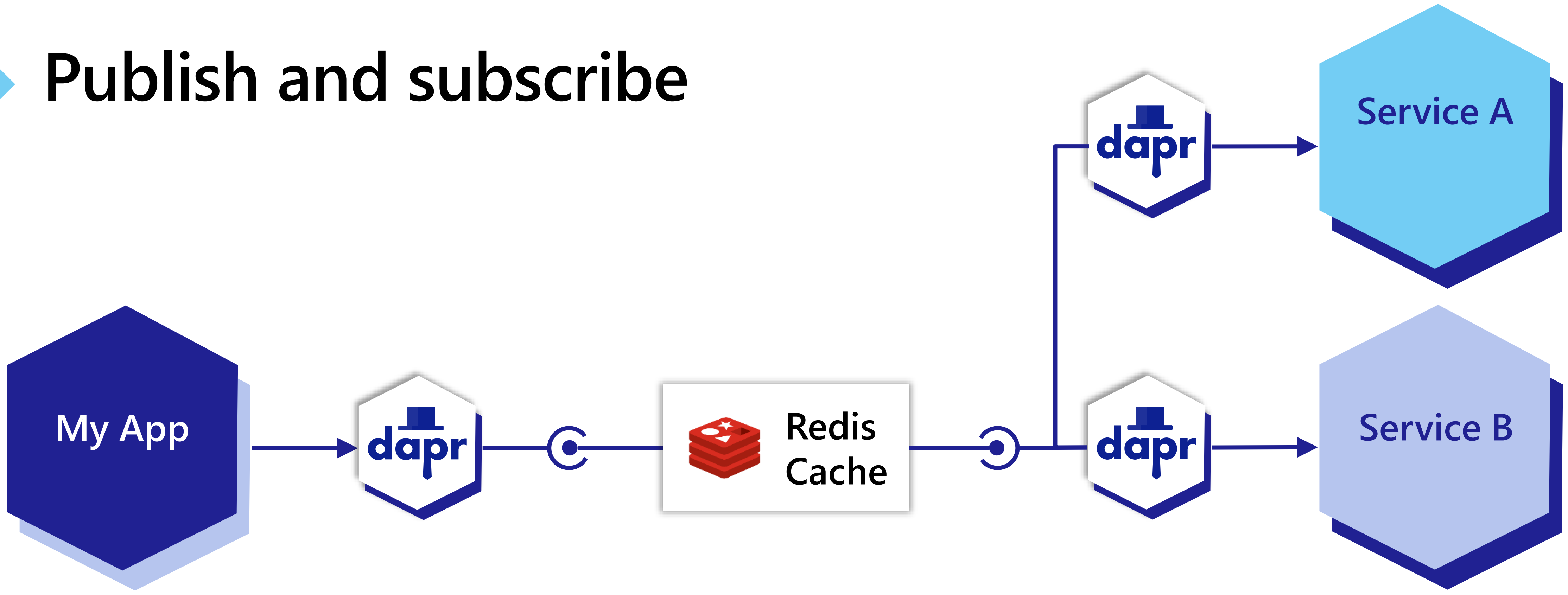
POST
`http://localhost:3500/v1.0/invoke/servicea/method/neworder`

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

POST
`http://10.0.0.2:8000/neworder`

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

Publish and subscribe



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

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

POST
`http://10.0.0.2:8000/orders`
`http://10.0.0.4:8000/factory/orders`

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

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
```

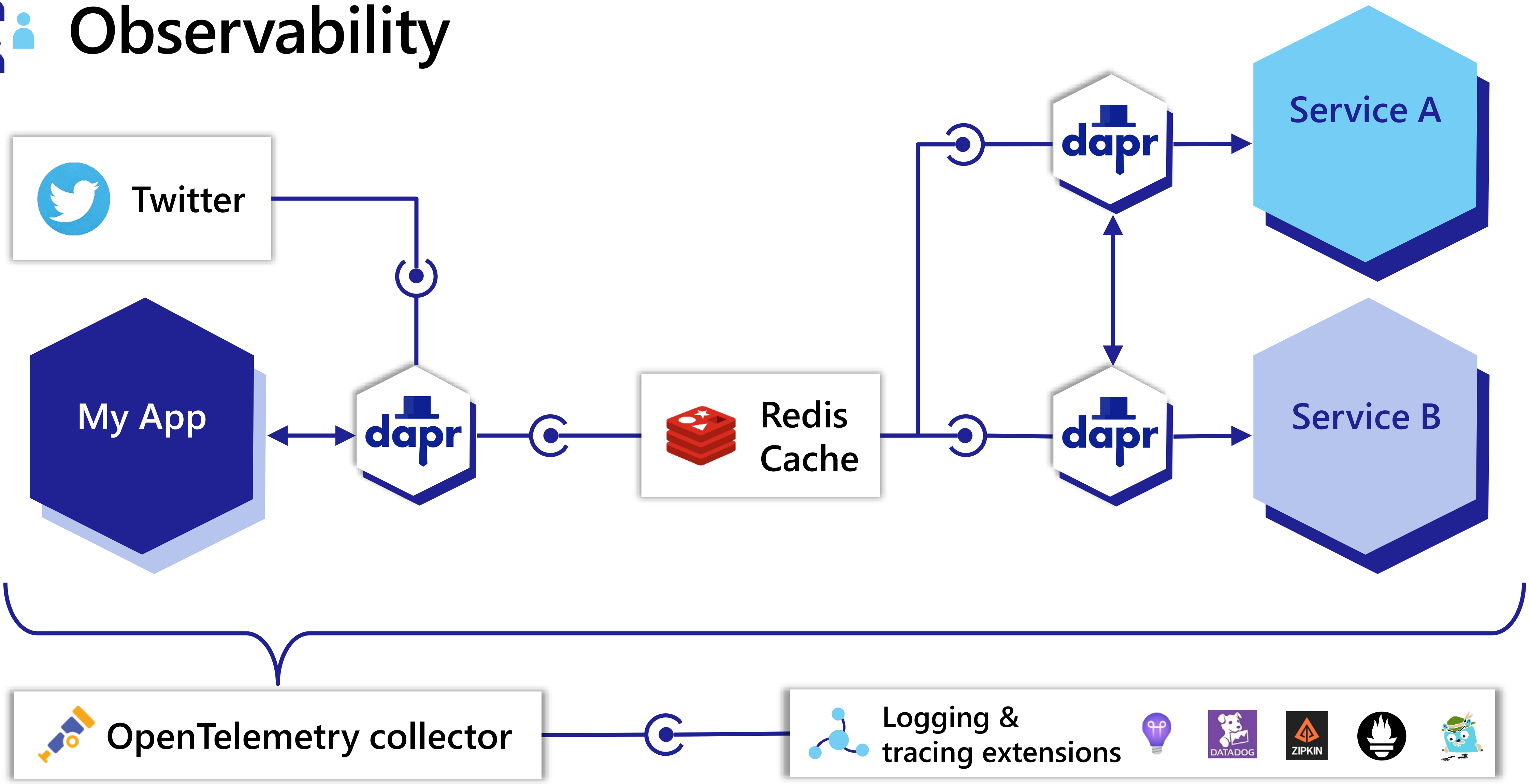
Publish to app

```
POST /order-processing
```

orders.yaml

```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"
```

Observability

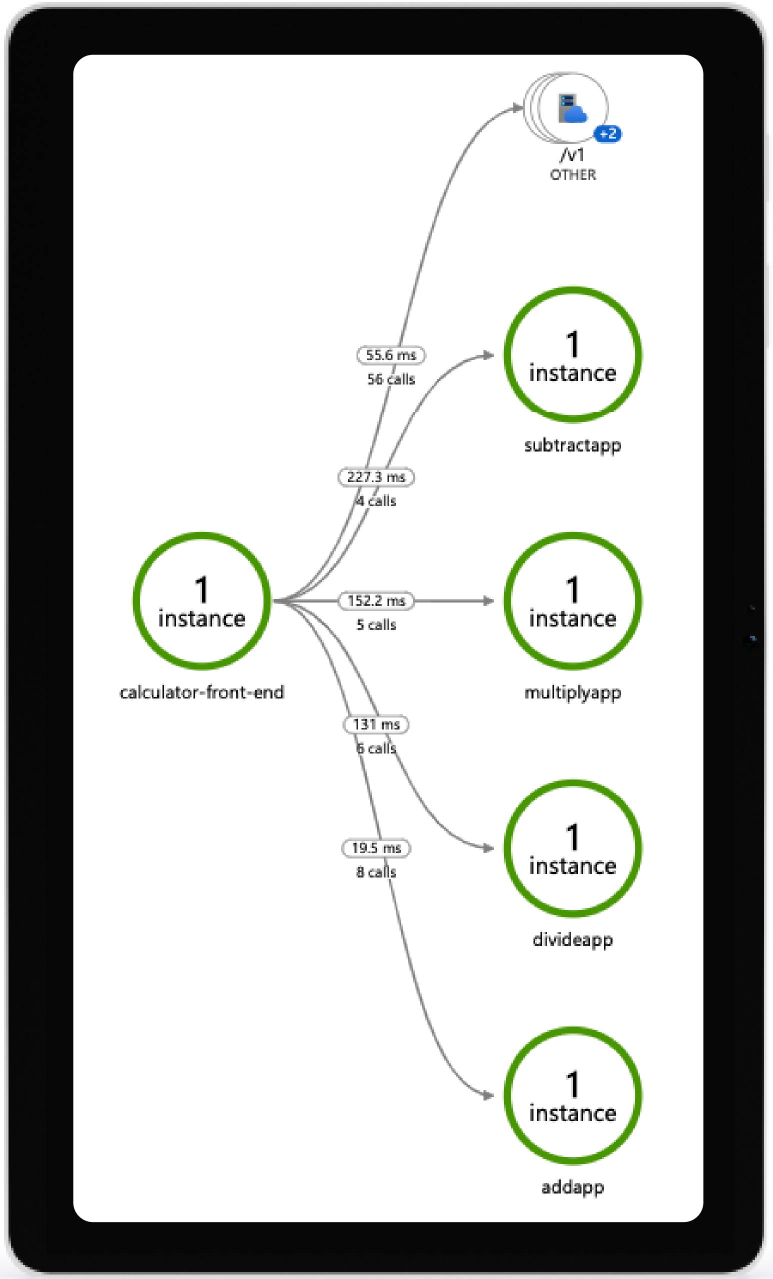


Distributed tracing

Emit tracing data from calls to/from Dapr sidecars and system services for easy application-level instrumentation

Dapr distributed tracing features:

- ✓ Built-in Zipkin collection and viewer
- ✓ Configurable sampling rates

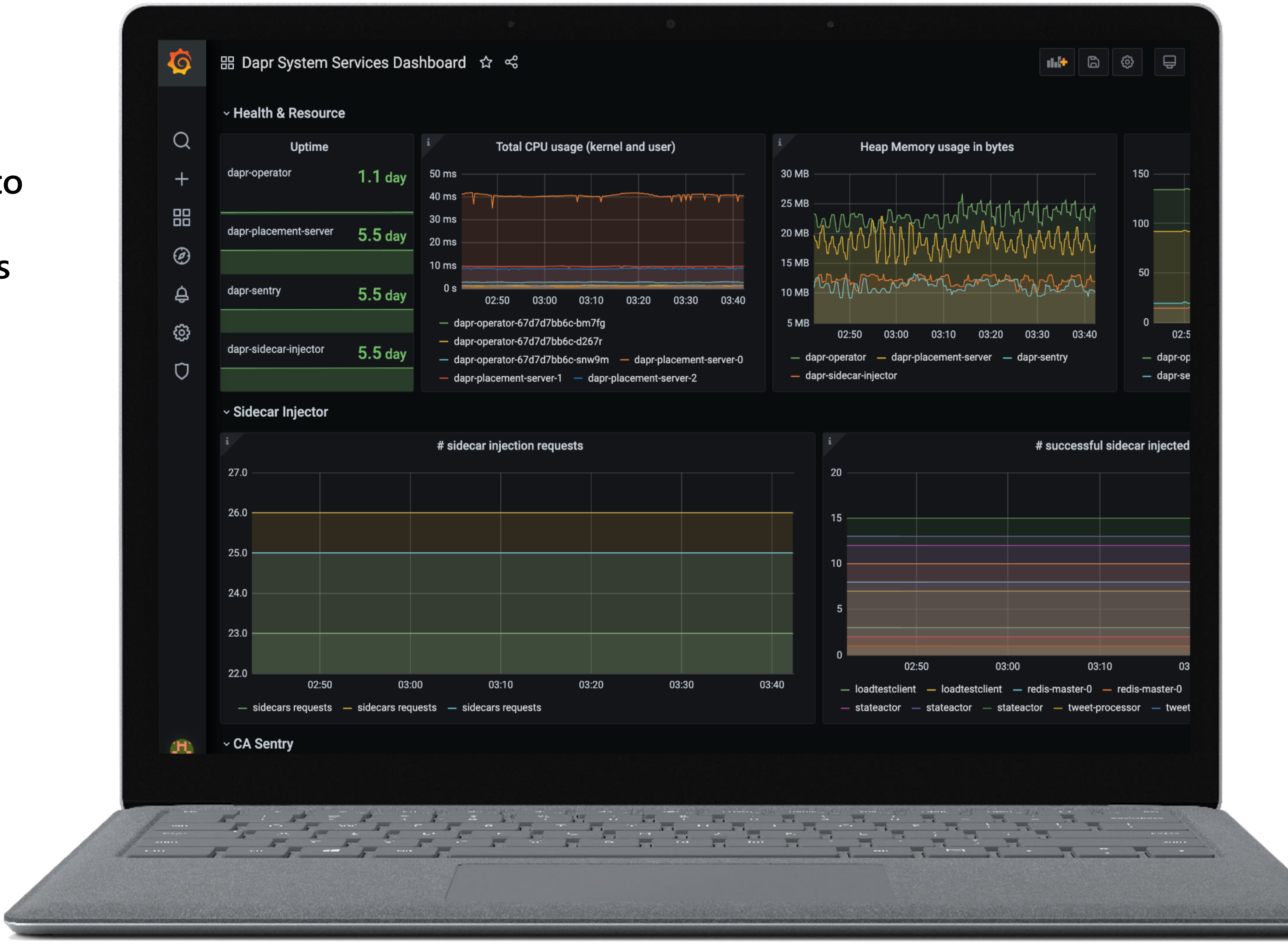


Metrics

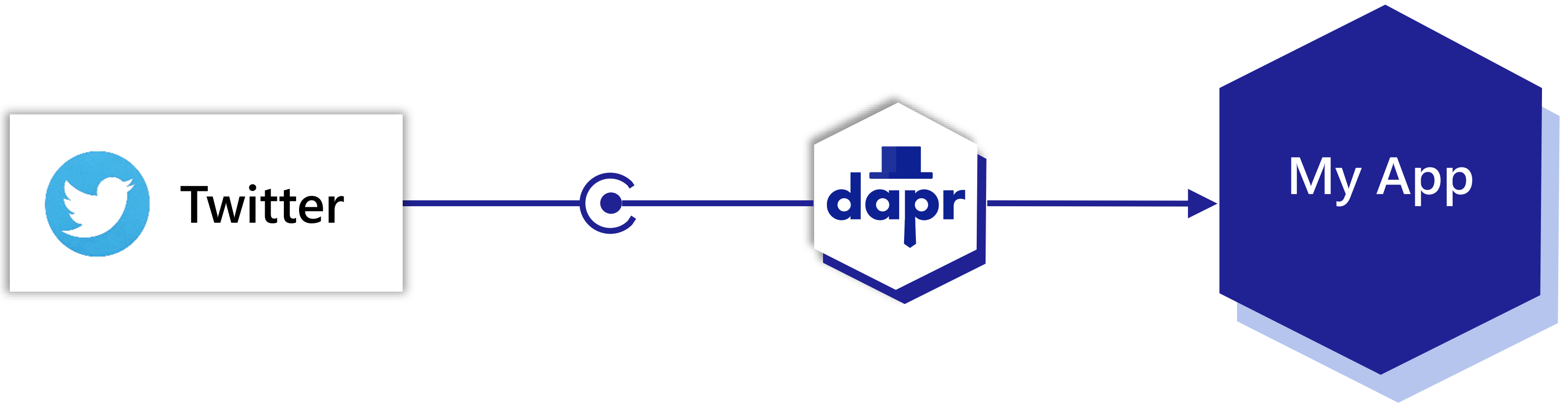
Built-in monitoring capabilities to understand the behavior of the Dapr sidecar and system services

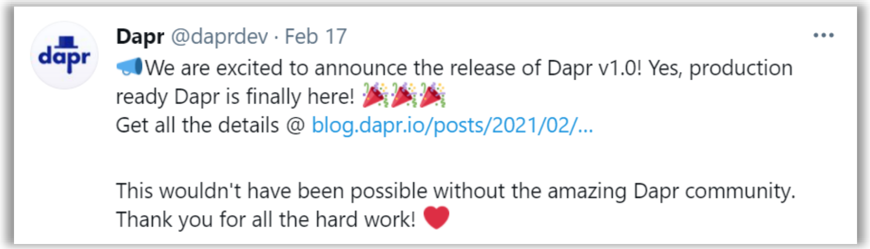
Dapr Metrics features:

- ✓ Call latency
- ✓ CPU/memory usage
- ✓ Error rates
- ✓ Sidecar injection failures
- ✓ System health


Dapr System Services Dashboard
Health & Resource
Uptime
dapr-operator 1.1 day
dapr-placement-server 5.5 day
dapr-sentry 5.5 day
dapr-sidecar-injector 5.5 day
Total CPU usage (kernel and user)
Heap Memory usage in bytes
Sidecar Injector
# sidecar injection requests
# successful sidecar injected
CA Sentry

Input triggers



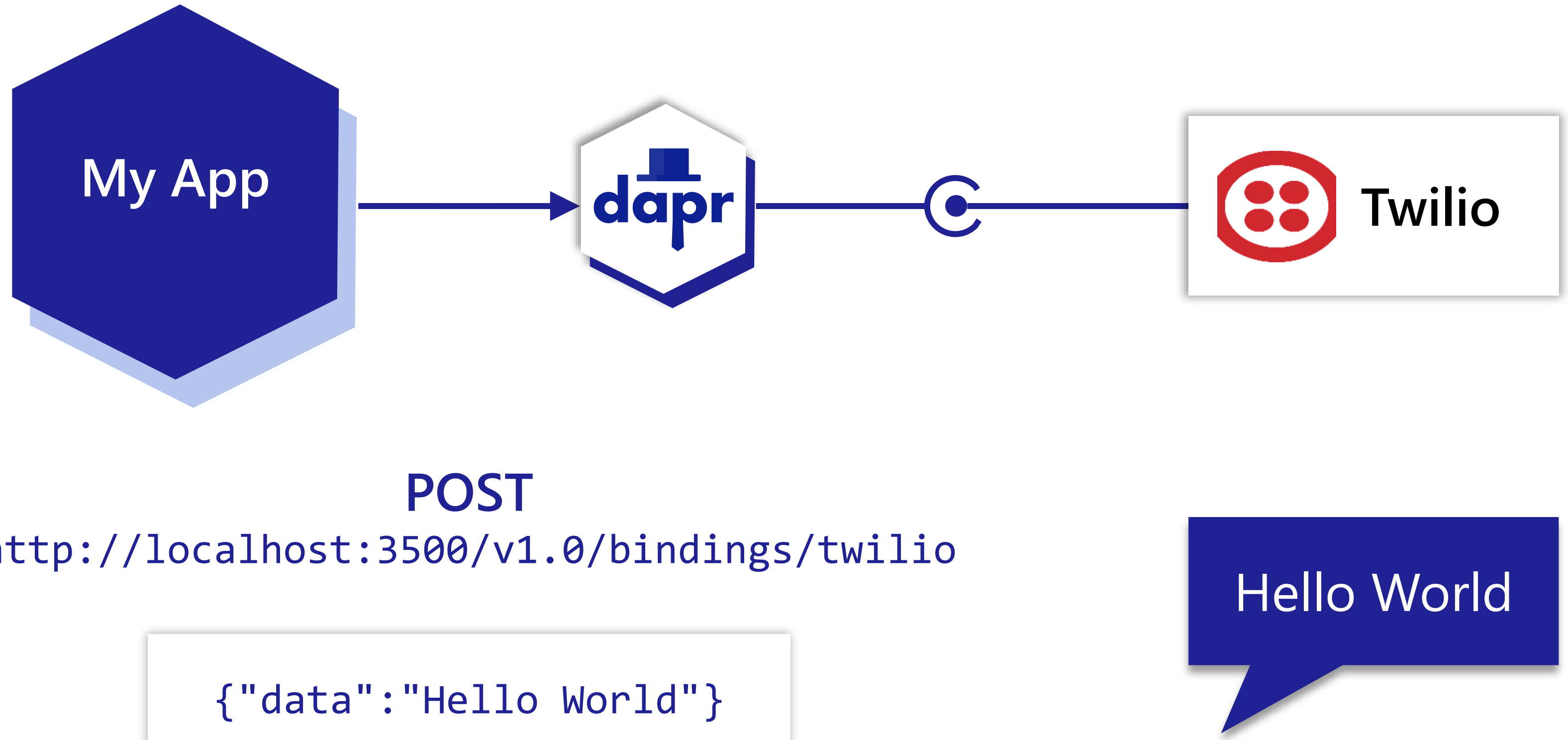

Dapr @daprdev · Feb 17
📣We are excited to announce the release of Dapr v1.0! Yes, production ready Dapr is finally here! 🎉🎉🎉
Get all the details @ blog.dapr.io/posts/2021/02/...

This wouldn't have been possible without the amazing Dapr community. Thank you for all the hard work! ❤️

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"}
```

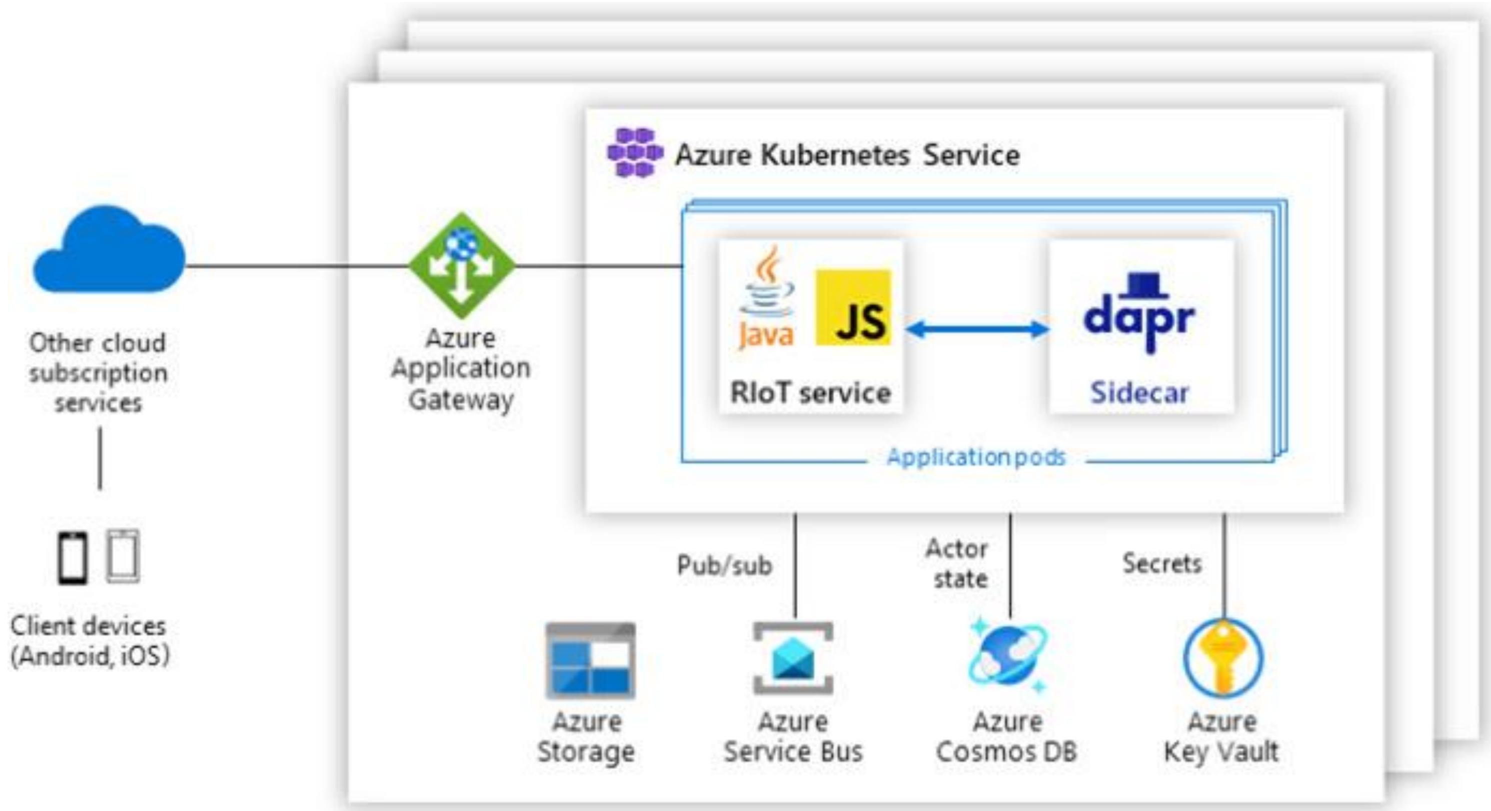
Case Study: Bosch

BOSCH
Invented for life

“We used the Dapr SDK to publish events, which gave us the time to focus on what we wanted to communicate, instead of, ‘How do I get this communication implemented?’ We could focus on the real things that matter.”

—Jasper Mang: Lead Developer
RIoT

- Residential IoT Services GmbH (RIoT) group chartered to implement and operate a smart home ecosystem
- Multi-language support for Java & JS
- Dapr eases the move to event-driven microservices

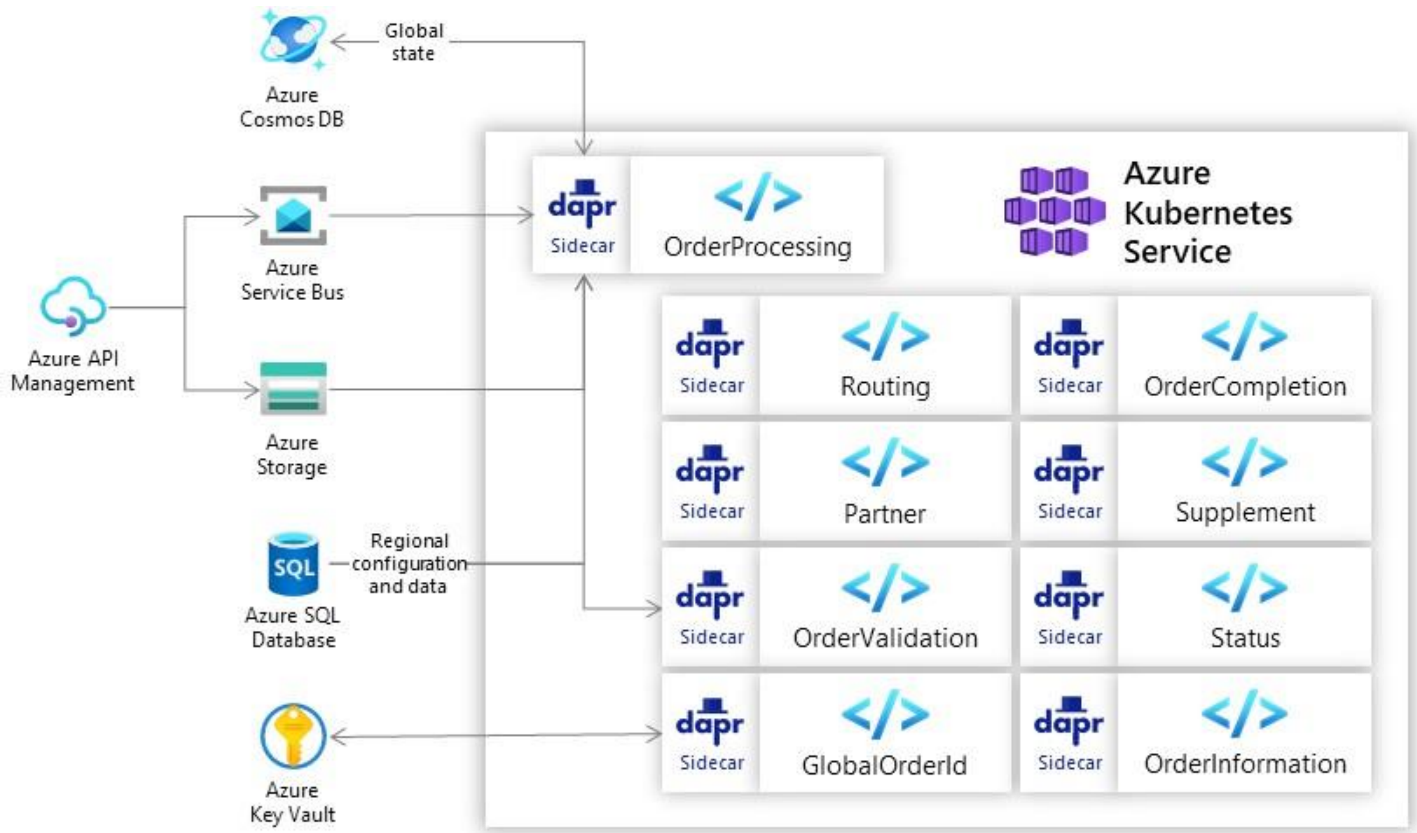


Case Study: Zeiss

- Microservice architecture for Order Processing application
- Support for multiple cloud and on-premises environments

"We wanted a platform-agnostic, microservices architecture with a very small footprint. We got that with Dapr and Azure Kubernetes Service."

—Kai Walter: Lead Architect
ZEISS



Dapr Demo

Dapr + Rancher K3S

Rancher K3S

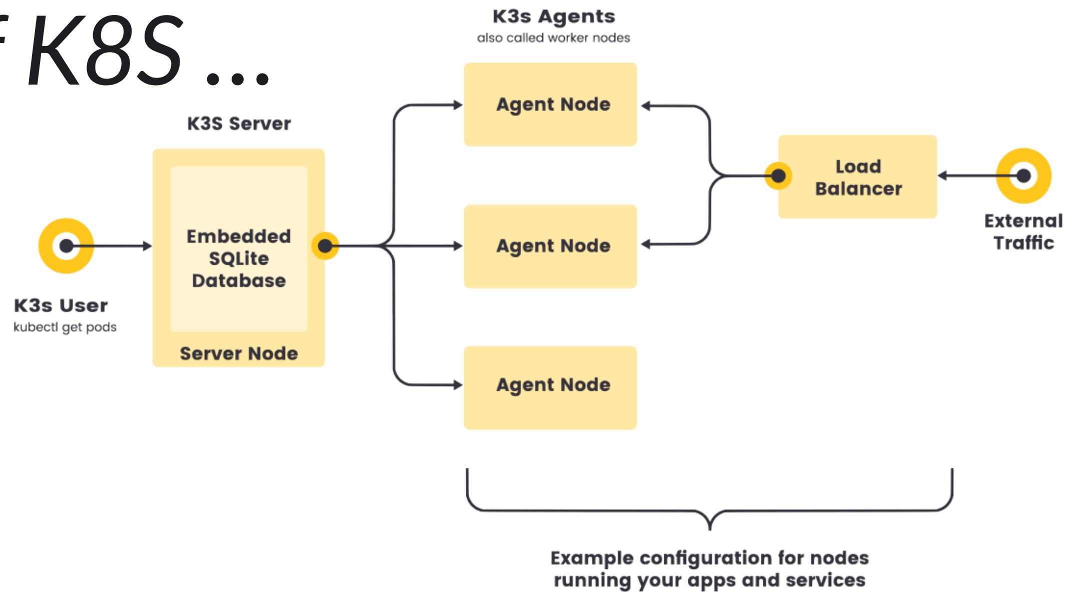
half the size (memory footprint) of K8S …

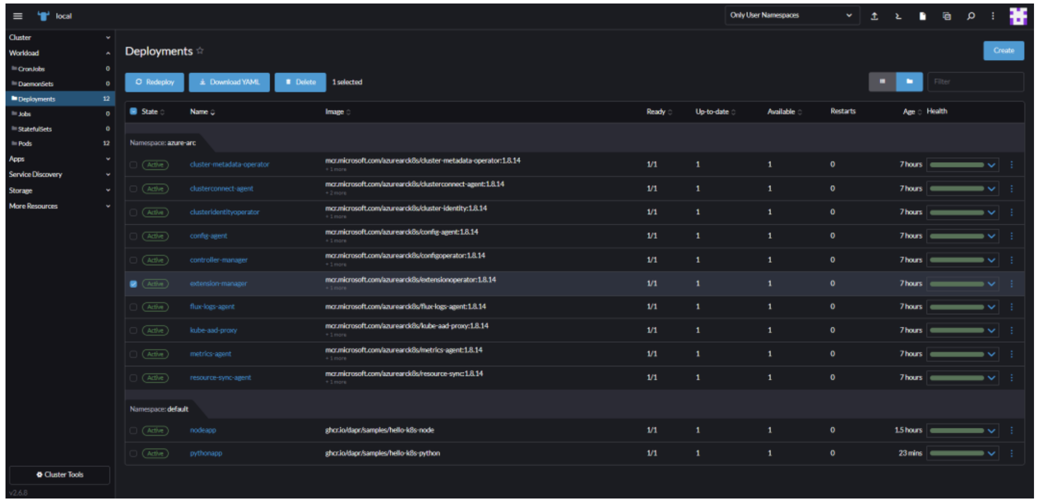
Lightweight Kubernetes. Easy to install, half the memory, all in a binary of less than 100 MB

Great for:

- Edge, IoT, CI, Development, ARM, Embedding K8s …

K3s is a fully compliant Kubernetes distribution with lightweight storage, packaged as a single binary, external dependencies minimized …



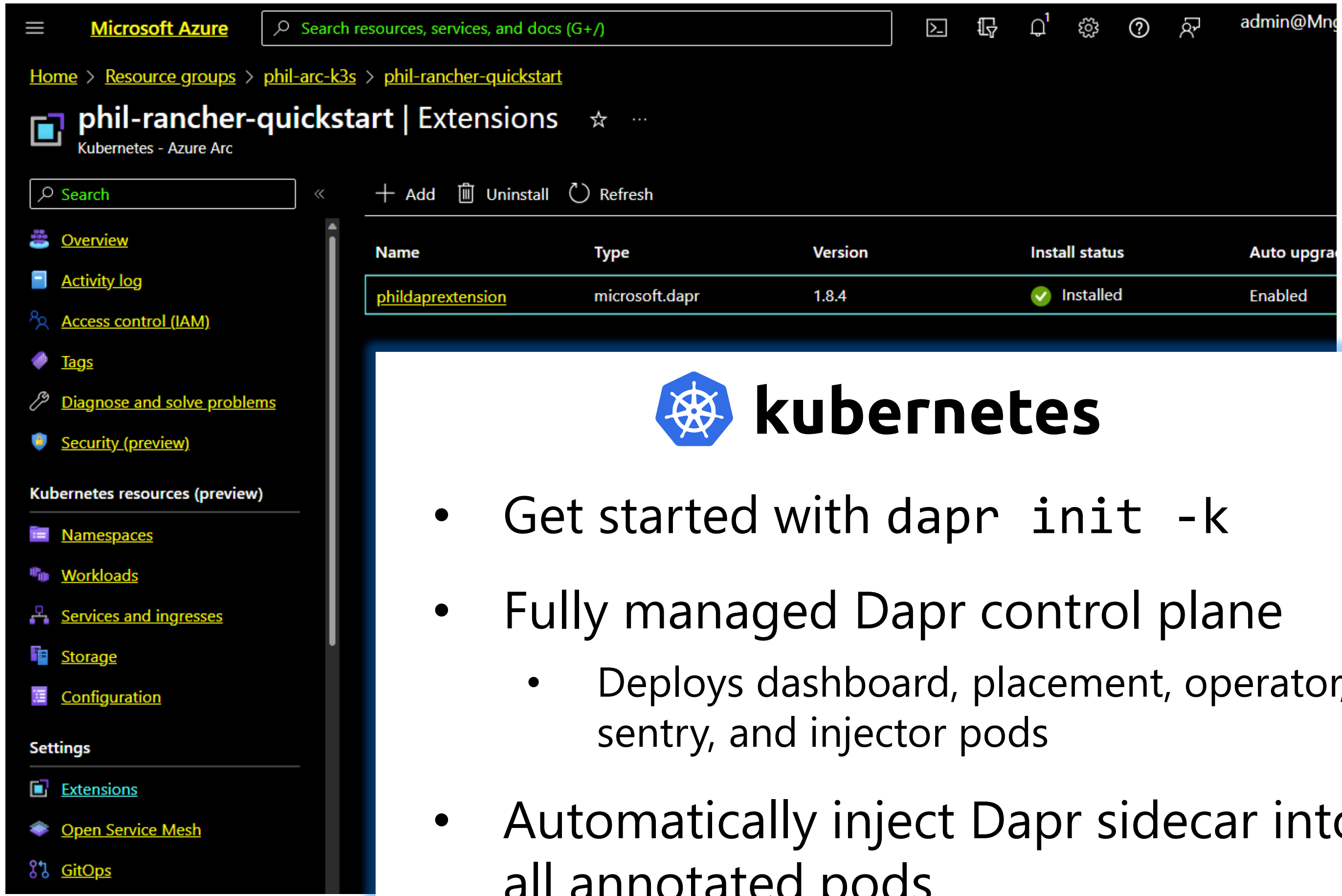


Deployments

State	Name	Image	Ready	Up-to-date	Available	Restarts	Age	Health
Namespace: azure-arc								
Active	cluster-metadata-operator	mcr.microsoft.com/azurearck8s/cluster-metadata-operator:1.8.14 + 1 more	1/1	1	1	0	7 hours	
Active	clusterconnect-agent	mcr.microsoft.com/azurearck8s/clusterconnect-agent:1.8.14 + 2 more	1/1	1	1	0	7 hours	
Active	clusteridentityoperator	mcr.microsoft.com/azurearck8s/cluster-identity:1.8.14 + 1 more	1/1	1	1	0	7 hours	
Active	config-agent	mcr.microsoft.com/azurearck8s/config-agent:1.8.14 + 1 more	1/1	1	1	0	7 hours	
Active	controller-manager	mcr.microsoft.com/azurearck8s/configoperator:1.8.14 + 1 more	1/1	1	1	0	7 hours	
Active	extension-manager	mcr.microsoft.com/azurearck8s/extensionoperator:1.8.14 + 1 more	1/1	1	1	0	7 hours	
Active	flux-logs-agent	mcr.microsoft.com/azurearck8s/flux-logs-agent:1.8.14	1/1	1	1	0	7 hours	
Active	kube-aad-proxy	mcr.microsoft.com/azurearck8s/kube-aad-proxy:1.8.14 + 1 more	1/1	1	1	0	7 hours	
Active	metrics-agent	mcr.microsoft.com/azurearck8s/metrics-agent:1.8.14 + 1 more	1/1	1	1	0	7 hours	
Active	resource-sync-agent	mcr.microsoft.com/azurearck8s/resource-sync:1.8.14 + 1 more	1/1	1	1	0	7 hours	
Namespace: default								
Active	nodeapp	ghcr.io/dapr/samples/hello-k8s-node	1/1	1	1	0	1.5 hours	
Active	pythonapp	ghcr.io/dapr/samples/hello-k8s-python	1/1	1	1	0	23 mins	

Dapr Installation and Version management

- Install with Dapr CLI
- Install with Helm
  - Azure Arc - Manage your Kubernetes with Extension framework



Microsoft Azure   Search resources, services, and docs (G+/)   admin@Mn

Home > Resource groups > phil-arc-k3s > phil-rancher-quickstart

phil-rancher-quickstart | Extensions

Kubernetes - Azure Arc

Search

+ Add   Uninstall   Refresh

- Overview
- Activity log
- Access control (IAM)
- Tags
- Diagnose and solve problems
- Security (preview)

Kubernetes resources (preview)

- Namespaces
- Workloads
- Services and ingresses
- Storage
- Configuration

Settings

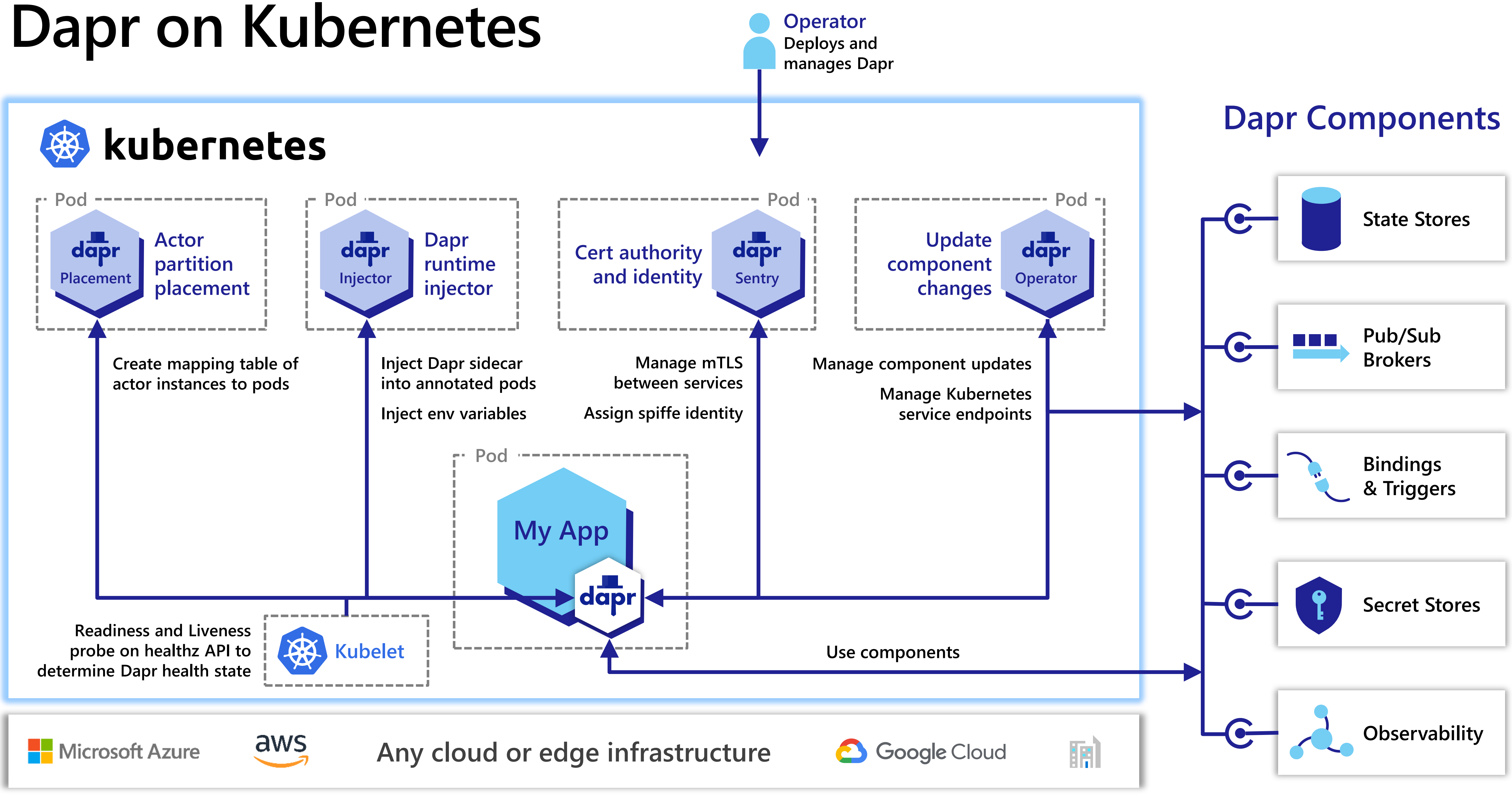
- Extensions
- Open Service Mesh
- GitOps

Name	Type	Version	Install status	Auto upgra
phildaprextension	microsoft.dapr	1.8.4	Installed	Enabled

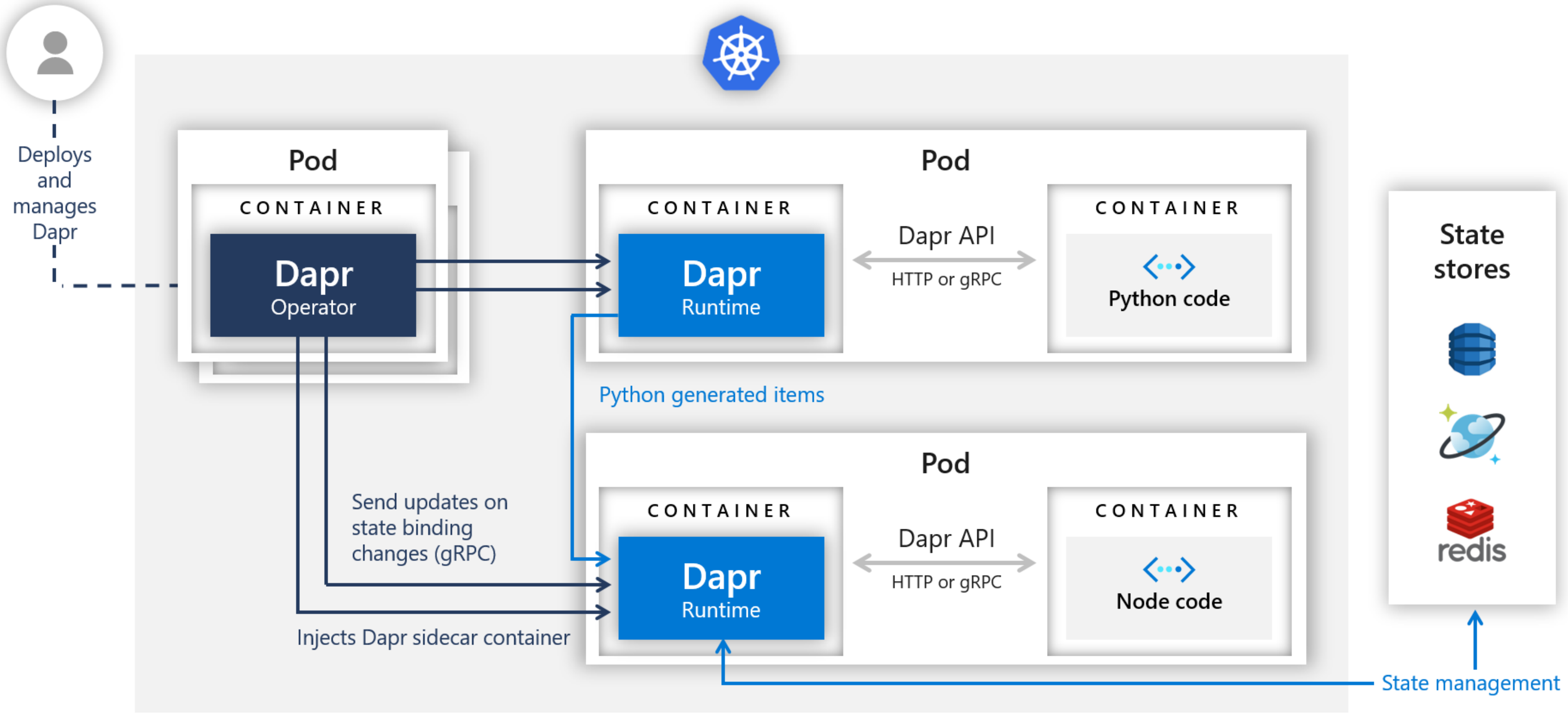
kubernetes

- 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

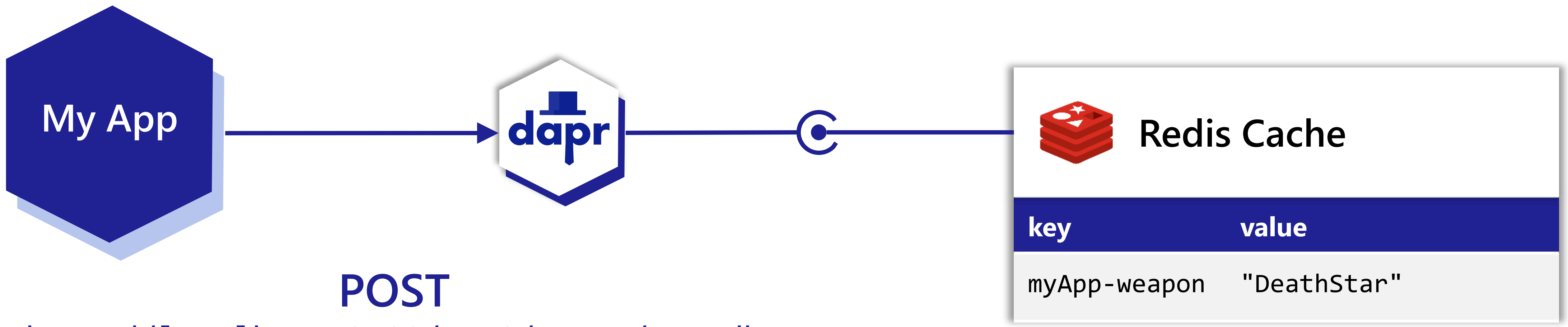
Dapr on Kubernetes



Dapr demo scenario – Hello Kubernetes



State management



key	value
myApp-weapon	"DeathStar"

POST

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

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

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
```

Node App Creation

dapr.io/enabled: true - this tells the Dapr control plane to inject a sidecar to this deployment

dapr.io/app-id: nodeapp - this assigns a unique ID or name to the Dapr application, so it can be sent messages to and communicated with by other Dapr apps

python.yaml

```
kind: Service
apiVersion: v1
metadata:
  name: nodeapp
  labels:
    app: node
spec:
  selector:
           …
  type: LoadBalancer
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nodeapp
  labels:
    app: node
spec:
  replicas: 1
          ...
        app: node
      annotations:
        dapr.io/enabled: "true"
        dapr.io/app-id: "nodeapp"
        dapr.io/app-port: "3000"
        dapr.io/enable-api-logging: "true"
```

Python App Creation

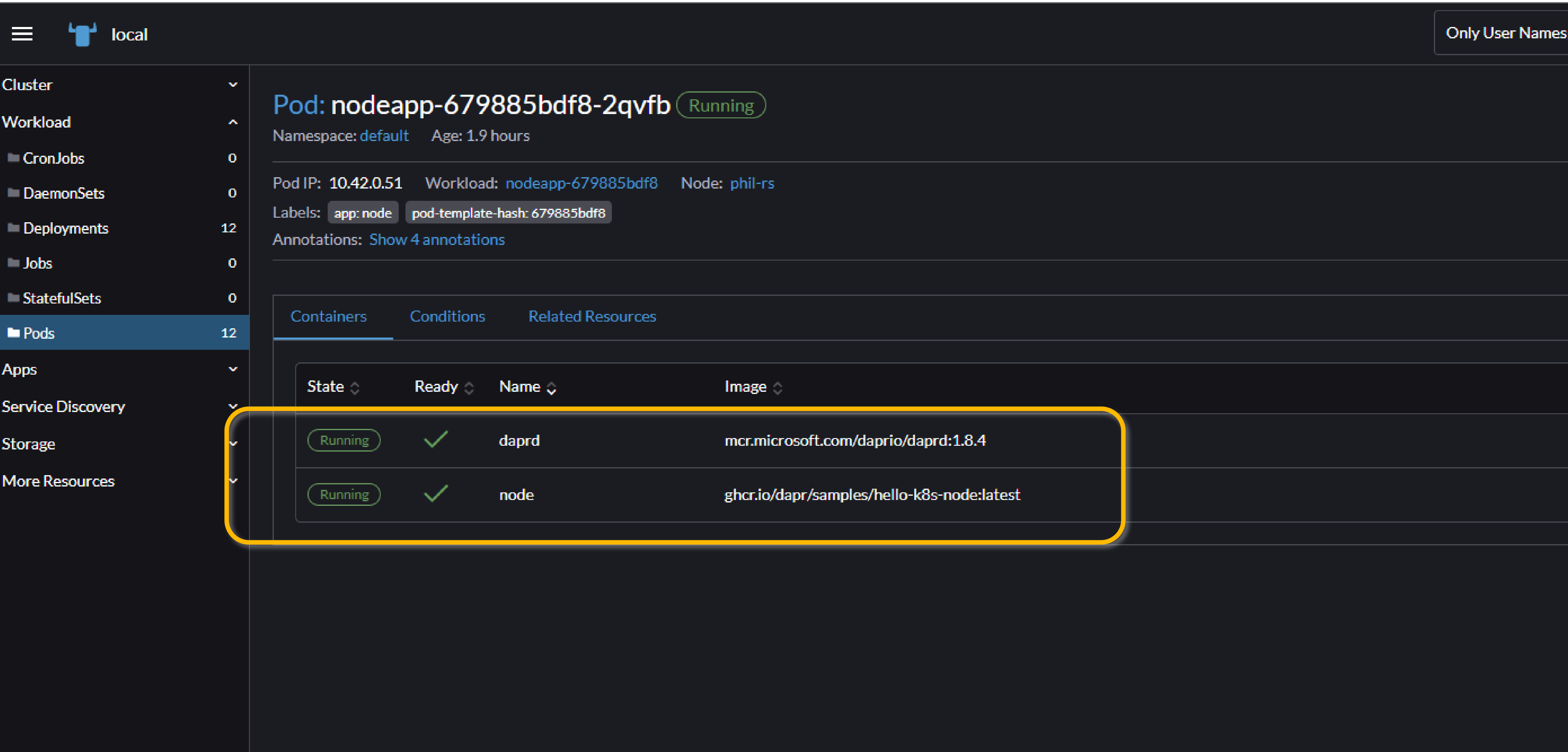
dapr.io/enabled: true - this tells the Dapr control plane to inject a sidecar to this deployment

dapr.io/app-id: pythonapp - this assigns a unique ID or name to the Dapr application, so it can be sent messages to and communicated with by other Dapr apps

python.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: pythonapp
  labels:
    app: python
spec:
  replicas: 1
  selector:
    matchLabels:
      app: python
  template:
    metadata:
      labels:
        app: python
      annotations:
        dapr.io/enabled: "true"
        dapr.io/app-id: "pythonapp"
        dapr.io/enable-api-logging: "true"
    spec:
      containers:
      - name: python
        image: ghcr.io/dapr/samples/hello-k8s-python:latest
```

Daprd as sidecar in your Pod



Pod: nodeapp-679885bdf8-2qvfb Running

Namespace: default Age: 1.9 hours

Pod IP: 10.42.0.51 Workload: nodeapp-679885bdf8 Node: phil-rs

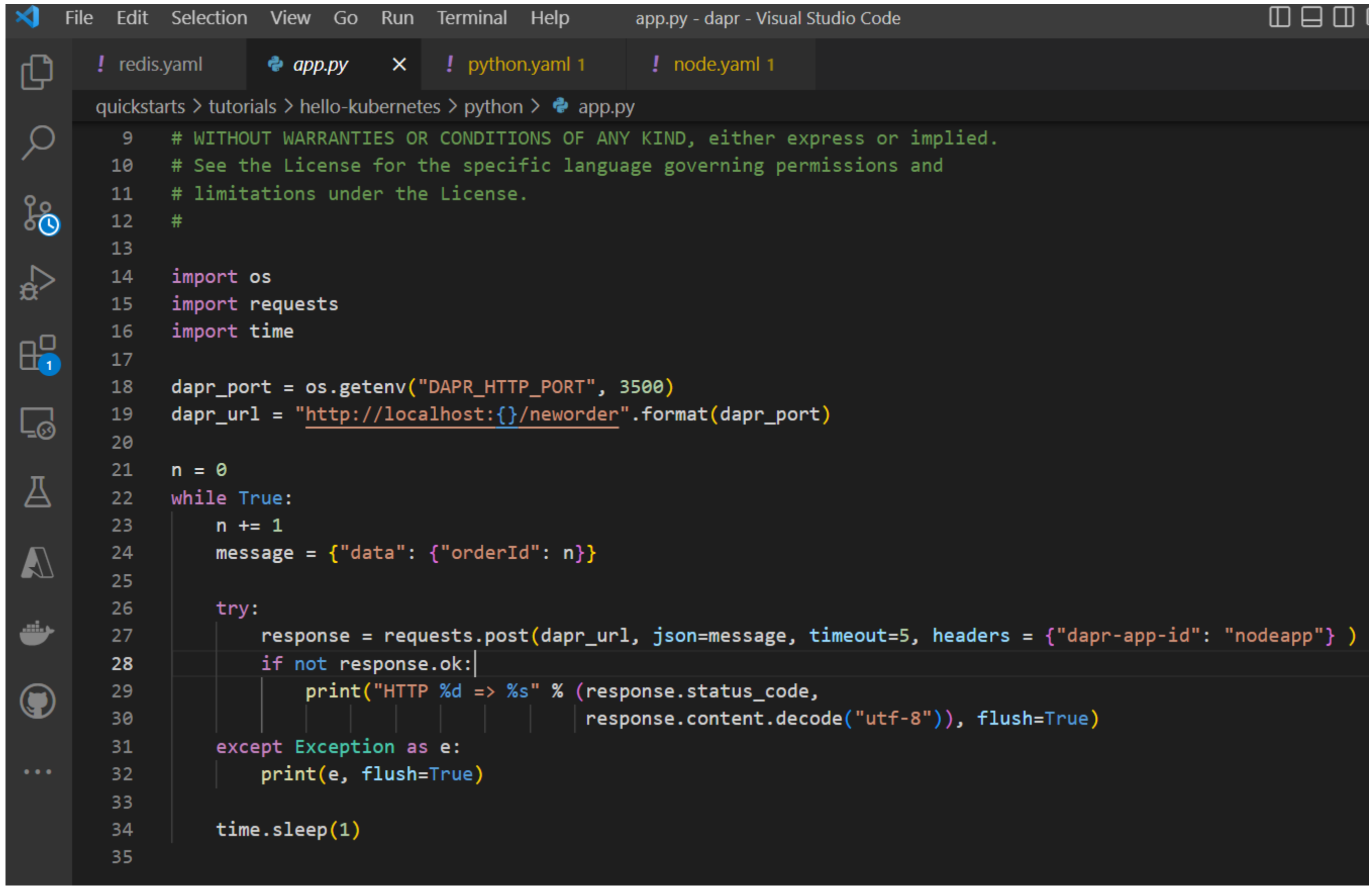
Labels: app: node pod-template-hash: 679885bdf8

Annotations: Show 4 annotations

State	Ready	Name	Image
Running	✓	daprd	mcr.microsoft.com/daprio/daprd:1.8.4
Running	✓	node	ghcr.io/dapr/samples/hello-k8s-node:latest

Python App

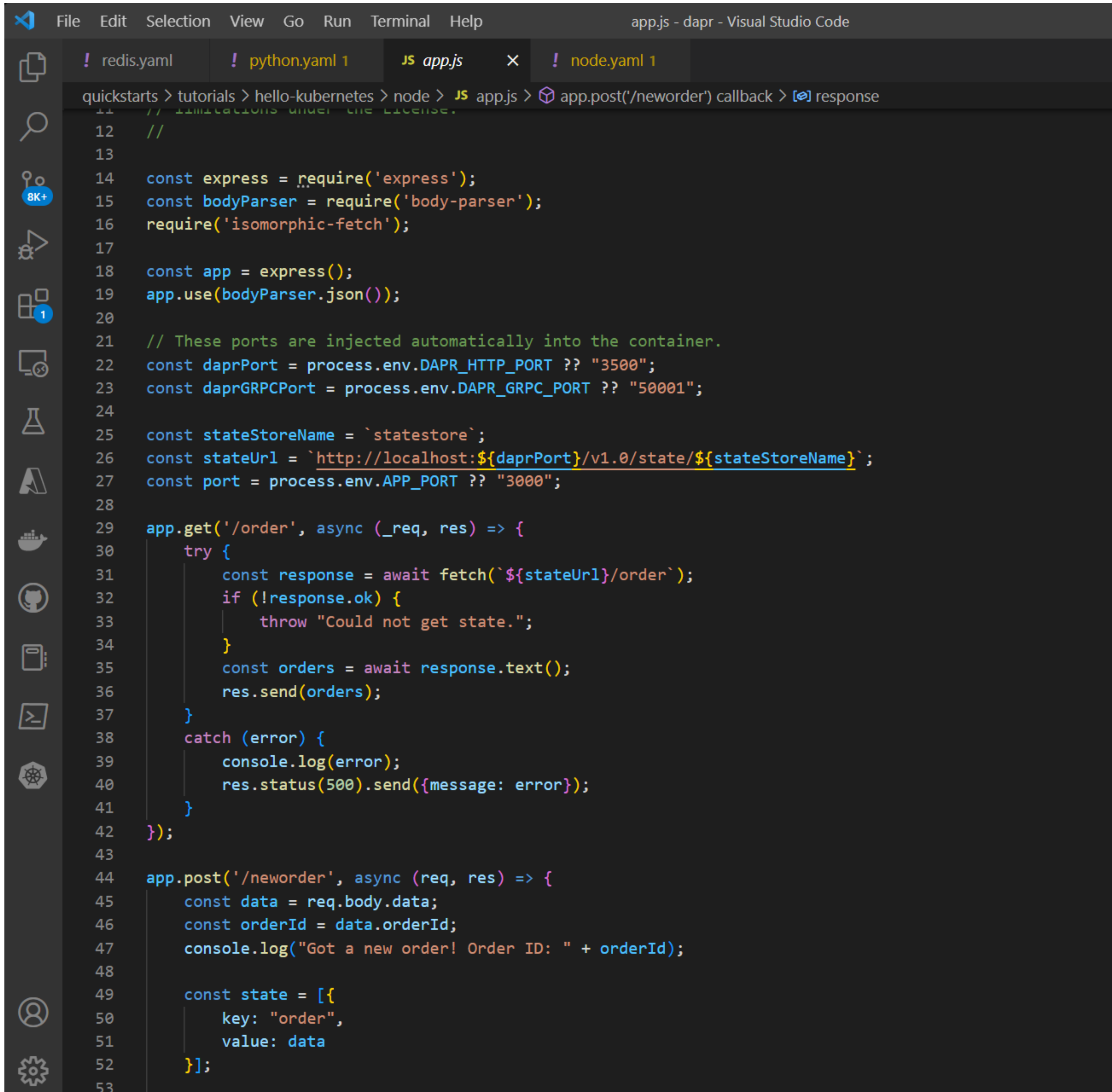
- This is a basic Python app that posts JSON messages to localhost:3500, which is the default listening port for Dapr
- Invoke the Node.js application's neworder endpoint by posting to v1.0/invoke/nodeapp/method/neworder.


File Edit Selection View Go Run Terminal Help
app.py - dapr - Visual Studio Code
redis.yaml
app.py
python.yaml 1
node.yaml 1
quickstarts > tutorials > hello-kubernetes > python > app.py

```python
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#

import os
import requests
import time

dapr_port = os.getenv("DAPR_HTTP_PORT", 3500)
dapr_url = "http://localhost:{}/neworder".format(dapr_port)

n = 0
while True:
    n += 1
    message = {"data": {"orderId": n}}

    try:
        response = requests.post(dapr_url, json=message, timeout=5, headers = {"dapr-app-id": "nodeapp"} )
        if not response.ok:
            print("HTTP %d => %s" % (response.status_code,
                                     response.content.decode("utf-8")), flush=True)
    except Exception as e:
        print(e, flush=True)

    time.sleep(1)
```

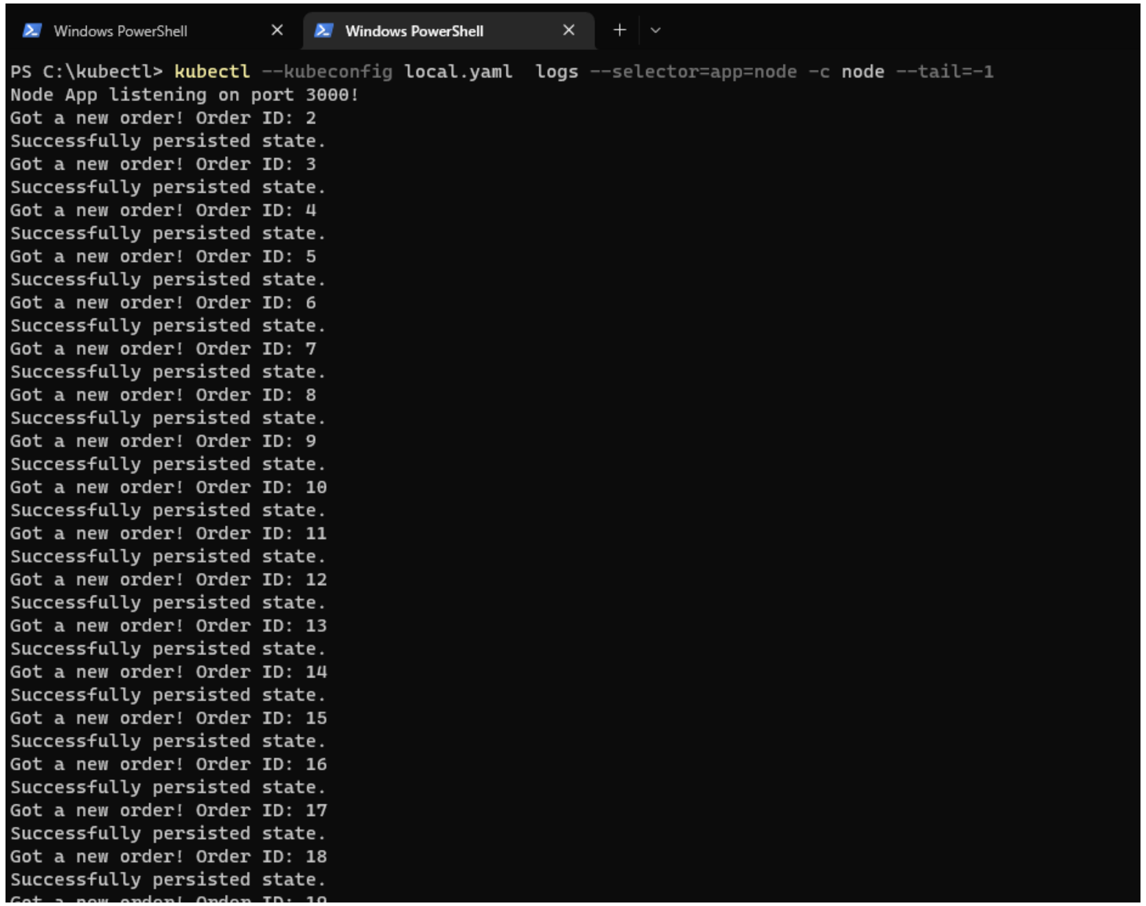
Node App

- Node App handling:
- /order: get latest order from statestore
- /neworder: get new order from Python App and persist in Redis statestore


File
Edit
Selection
View
Go
Run
Terminal
Help
app.js - dapr - Visual Studio Code
redis.yaml
python.yaml 1
app.js
node.yaml 1
quickstarts > tutorials > hello-kubernetes > node > app.js > app.post('/neworder') callback > response
8K+
1

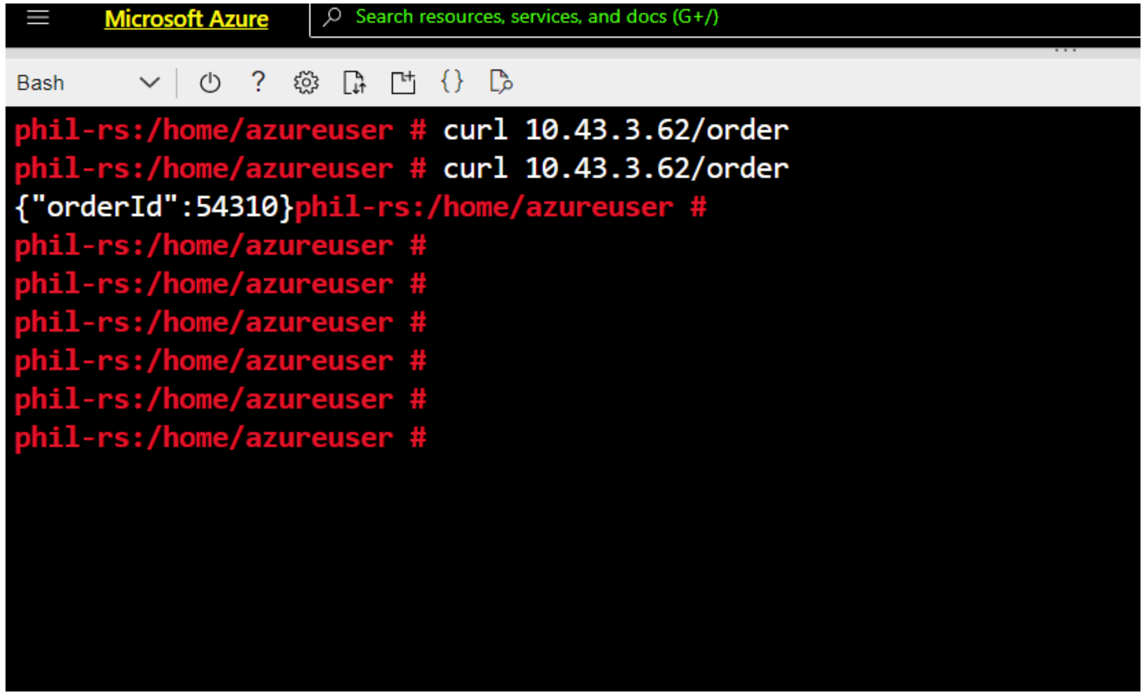
```javascript
// limitations under the License.
//

const express = require('express');
const bodyParser = require('body-parser');
require('isomorphic-fetch');

const app = express();
app.use(bodyParser.json());

// These ports are injected automatically into the container.
const daprPort = process.env.DAPR_HTTP_PORT ?? "3500";
const daprGRPCPort = process.env.DAPR_GRPC_PORT ?? "50001";

const stateStoreName = `statestore`;
const stateUrl = `http://localhost:${daprPort}/v1.0/state/${stateStoreName}`;
const port = process.env.APP_PORT ?? "3000";

app.get('/order', async (_req, res) => {
    try {
        const response = await fetch(`${stateUrl}/order`);
        if (!response.ok) {
            throw "Could not get state.";
        }
        const orders = await response.text();
        res.send(orders);
    }
    catch (error) {
        console.log(error);
        res.status(500).send({message: error});
    }
});

app.post('/neworder', async (req, res) => {
    const data = req.body.data;
    const orderId = data.orderId;
    console.log("Got a new order! Order ID: " + orderId);

    const state = [{
        key: "order",
        value: data
    }];
```

Observe Node App Messages and Confirm state persistence



```
PS C:\kubectl> kubectl --kubeconfig local.yaml  logs --selector=app=node -c node --tail=-1
Node App listening on port 3000!
Got a new order! Order ID: 2
Successfully persisted state.
Got a new order! Order ID: 3
Successfully persisted state.
Got a new order! Order ID: 4
Successfully persisted state.
Got a new order! Order ID: 5
Successfully persisted state.
Got a new order! Order ID: 6
Successfully persisted state.
Got a new order! Order ID: 7
Successfully persisted state.
Got a new order! Order ID: 8
Successfully persisted state.
Got a new order! Order ID: 9
Successfully persisted state.
Got a new order! Order ID: 10
Successfully persisted state.
Got a new order! Order ID: 11
Successfully persisted state.
Got a new order! Order ID: 12
Successfully persisted state.
Got a new order! Order ID: 13
Successfully persisted state.
Got a new order! Order ID: 14
Successfully persisted state.
Got a new order! Order ID: 15
Successfully persisted state.
Got a new order! Order ID: 16
Successfully persisted state.
Got a new order! Order ID: 17
Successfully persisted state.
Got a new order! Order ID: 18
Successfully persisted state.
Got a new order! Order ID: 19
```

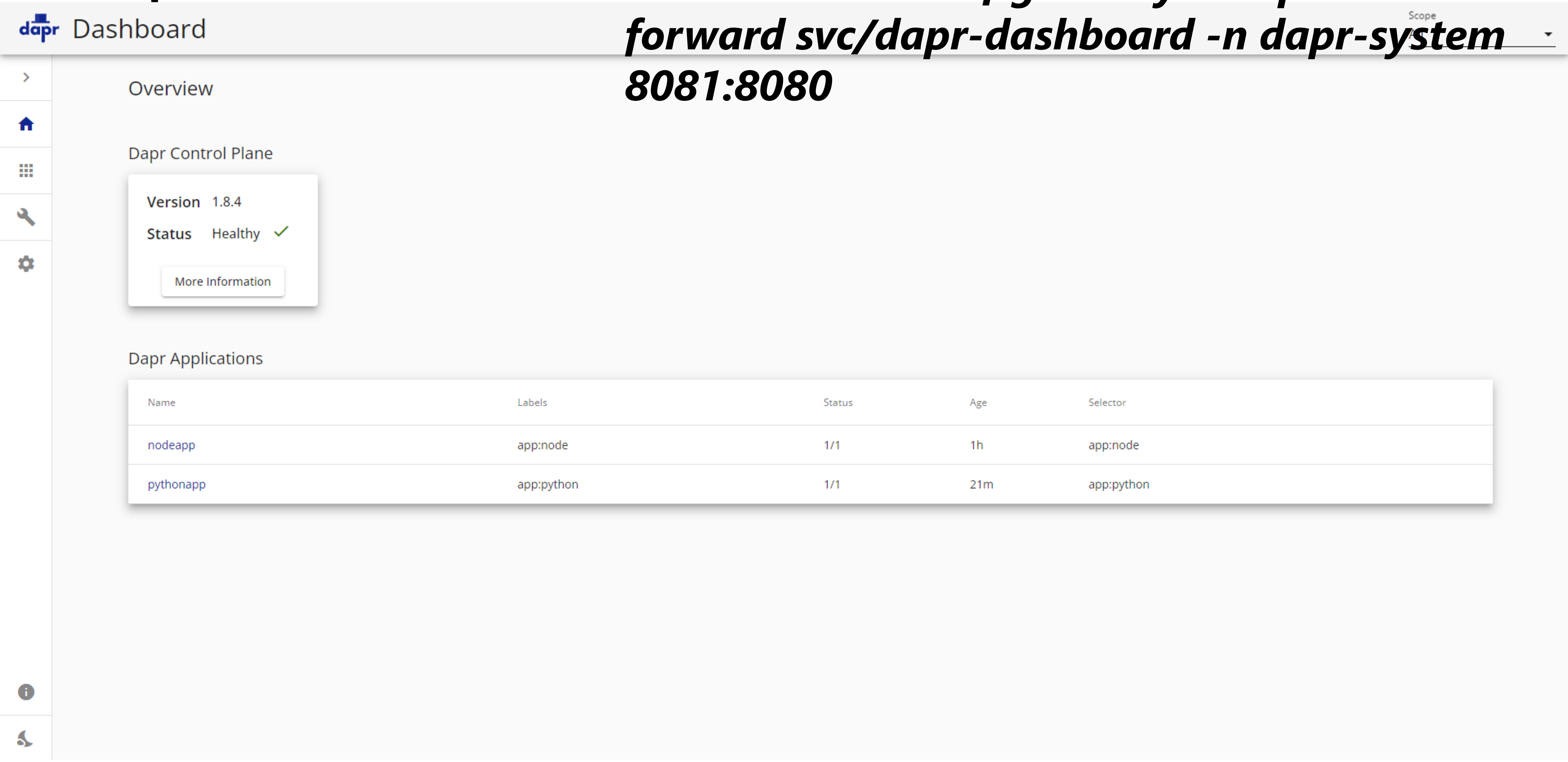
Last Order from Python App to Node App



```
phil-rs:/home/azureuser # curl 10.43.3.62/order
phil-rs:/home/azureuser # curl 10.43.3.62/order
{"orderId":54310}phil-rs:/home/azureuser #
phil-rs:/home/azureuser #
phil-rs:/home/azureuser #
phil-rs:/home/azureuser #
phil-rs:/home/azureuser #
phil-rs:/home/azureuser #
phil-rs:/home/azureuser #
```

Dapr Dashboard

kubectl --kubeconfig local.yaml port-forward svc/dapr-dashboard -n dapr-system 8081:8080



Overview

Dapr Control Plane

Version 1.8.4
Status Healthy ✓

Dapr Applications

Name	Labels	Status	Age	Selector
nodeapp	app:node	1/1	1h	app:node
pythonapp	app:python	1/1	21m	app:python

Azure Arc enabled Kubernetes

Connect, manage, and operate Kubernetes clusters and applications running anywhere using Azure Arc

Connect

Support for multiple flavors
Deploy to an existing cluster
OSS ecosystem friendly

Configure

Configure GitOps workflows
Enforce desired state across clusters
Cluster & Namespace support

Operate and Monitor

Azure Monitor Integration
Health status reporting
Cluster & App observability

Govern and Secure

Built-in Azure Policies
Cluster security baseline
Role-Based Access Control
Compliance across environments

Any infrastructure, any Kubernetes

kubeadm
AKS
OpenShift
EKS
GKE
VMware Tanzu

Dapr building blocks

Service-to-service invocation

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

State management

Create long running, stateless and stateful services

Publish and subscribe

Secure, scalable messaging between services

Bindings (input/output)

Trigger code through events from a large array of inputs

Input and output bindings to external resources including databases and queues

Actors

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

Observability

See and measure the message calls across components and networked services

Secrets

Securely access secrets from your application

Configuration

Manage and be notified of application configuration changes

Join Dapr Community

- dapr/community
- Communication
- Questions and issues
- Release planning meetings
  - Every Tuesday at 9 a.m. PST, one-hour release meetings
- Community meetings
  - Every two weeks, a community meeting to showcase new features, review upcoming milestones, and engage in a Q&A

Thank you!