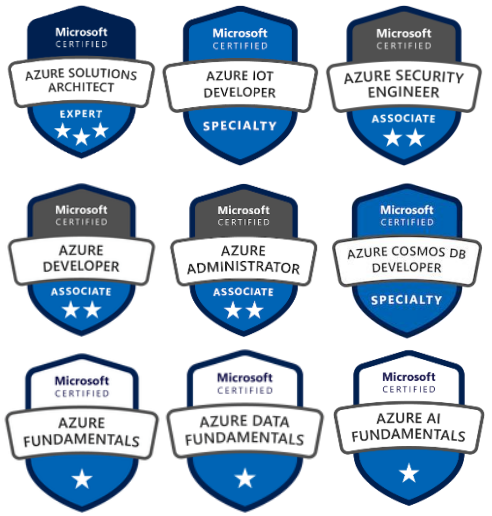
Dapr

Como o Dapr (Distributed Application Runtime) pode simplificar o desenvolvimento de aplicações em microservices

Walter Silvestre Coan

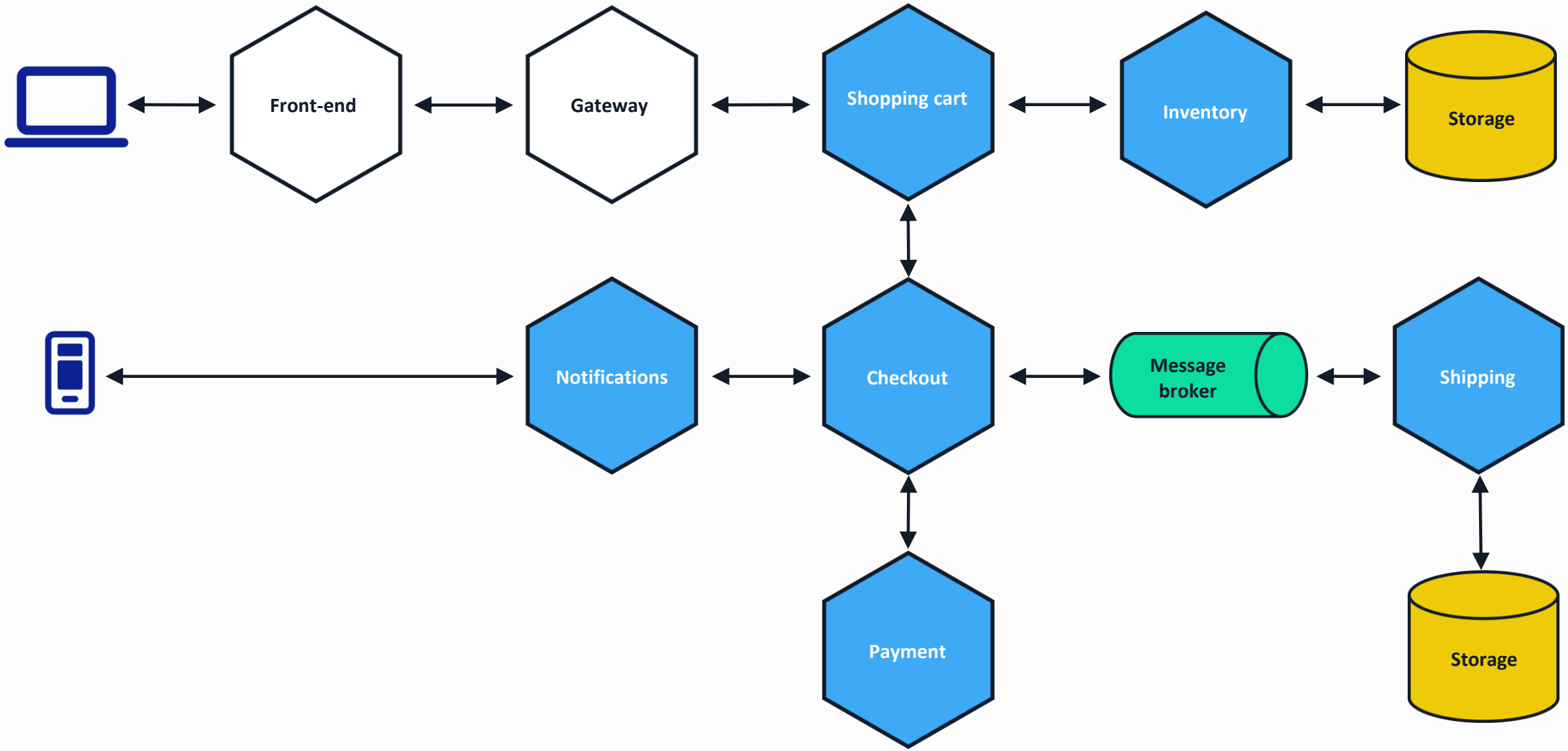
Walter Silvestre Coan

- Mestre em Ciência da Computação na área de Sistemas Distribuídos e Redes de Sensores sem Fio – PUCPR
- Cloud Solutions Architect na CDB Data Solution
- Microsoft Certified Trainer MCT na Ka Solution
- AWS Authorized Instructor na Ka Solution
- Professor do Bacharelado em Sistemas de Informação e do Bacharelado em Engenharia de Software da UNIVILLE
- Microsoft MVP em Azure e Internet das Coisas

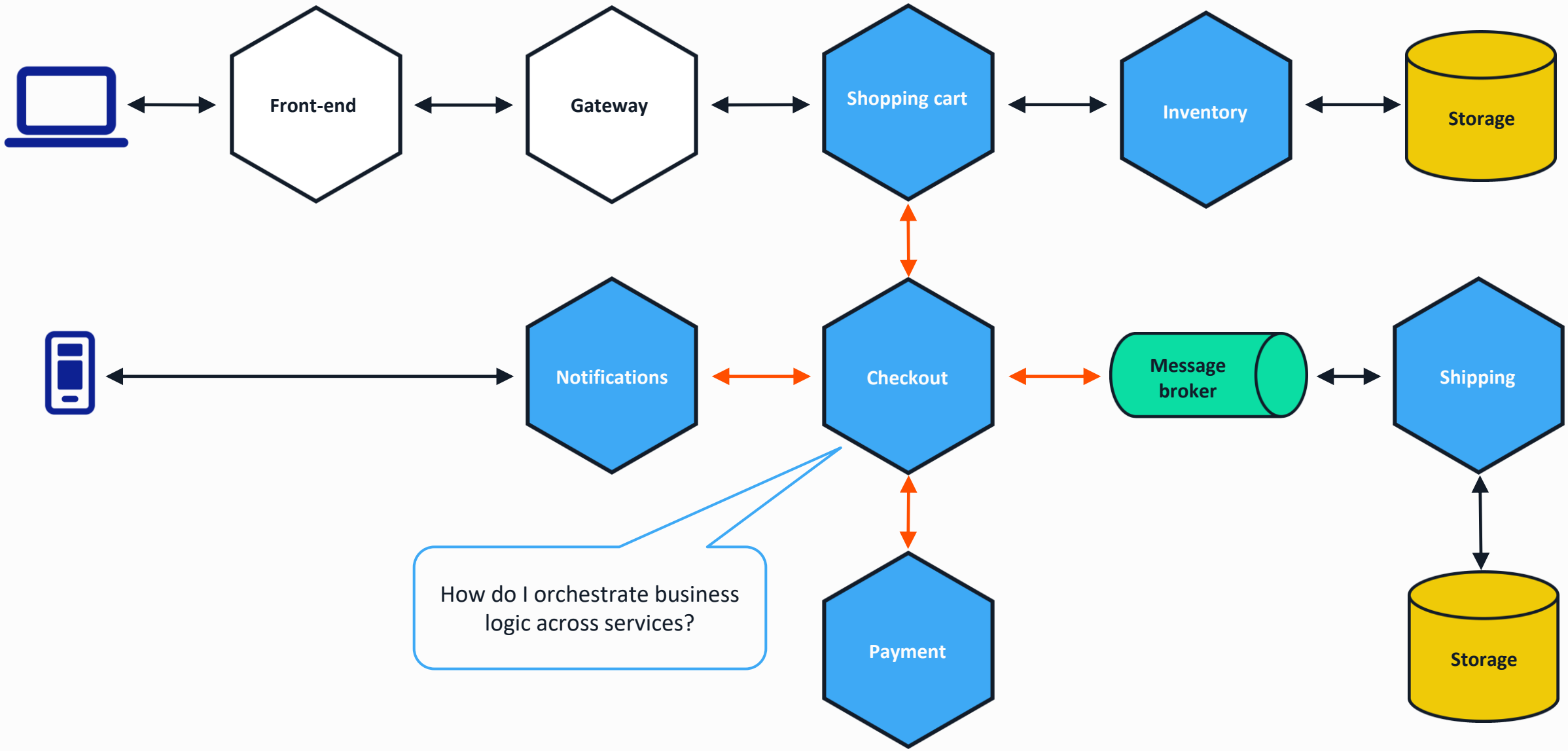


https://github.com/waltercoan/tdcflorianopolis2024-dapr

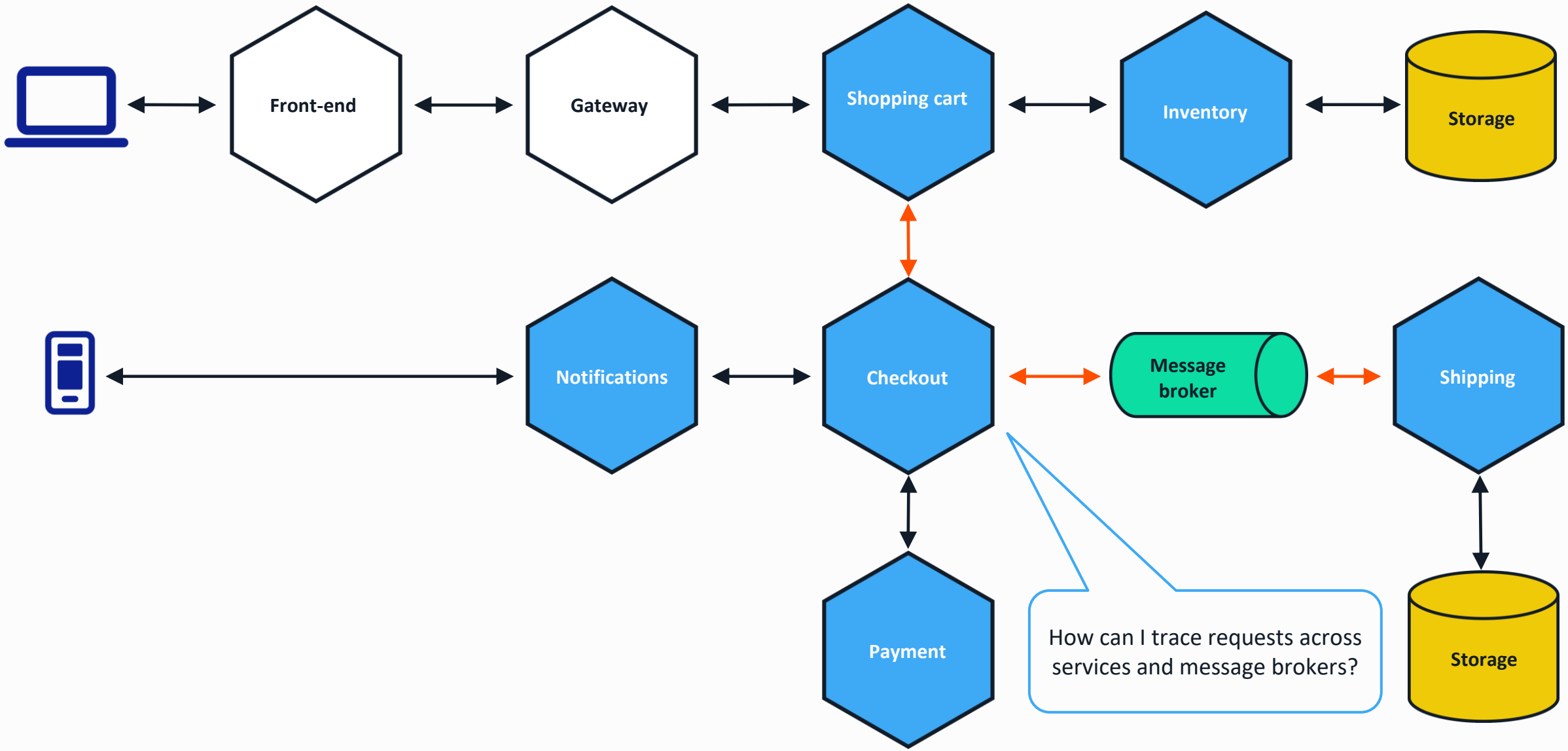
Distributed applications



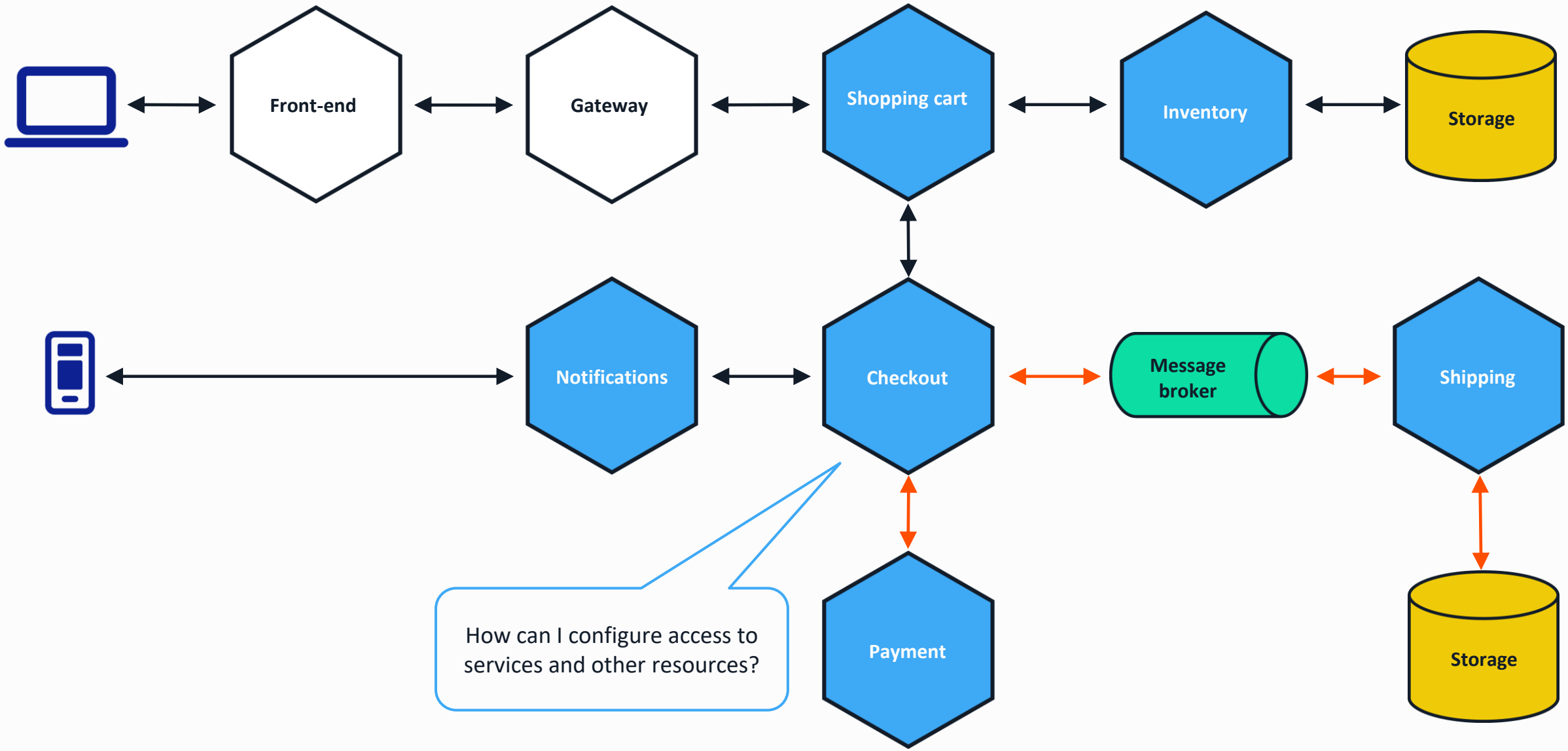
Developer challenges – Service orchestration



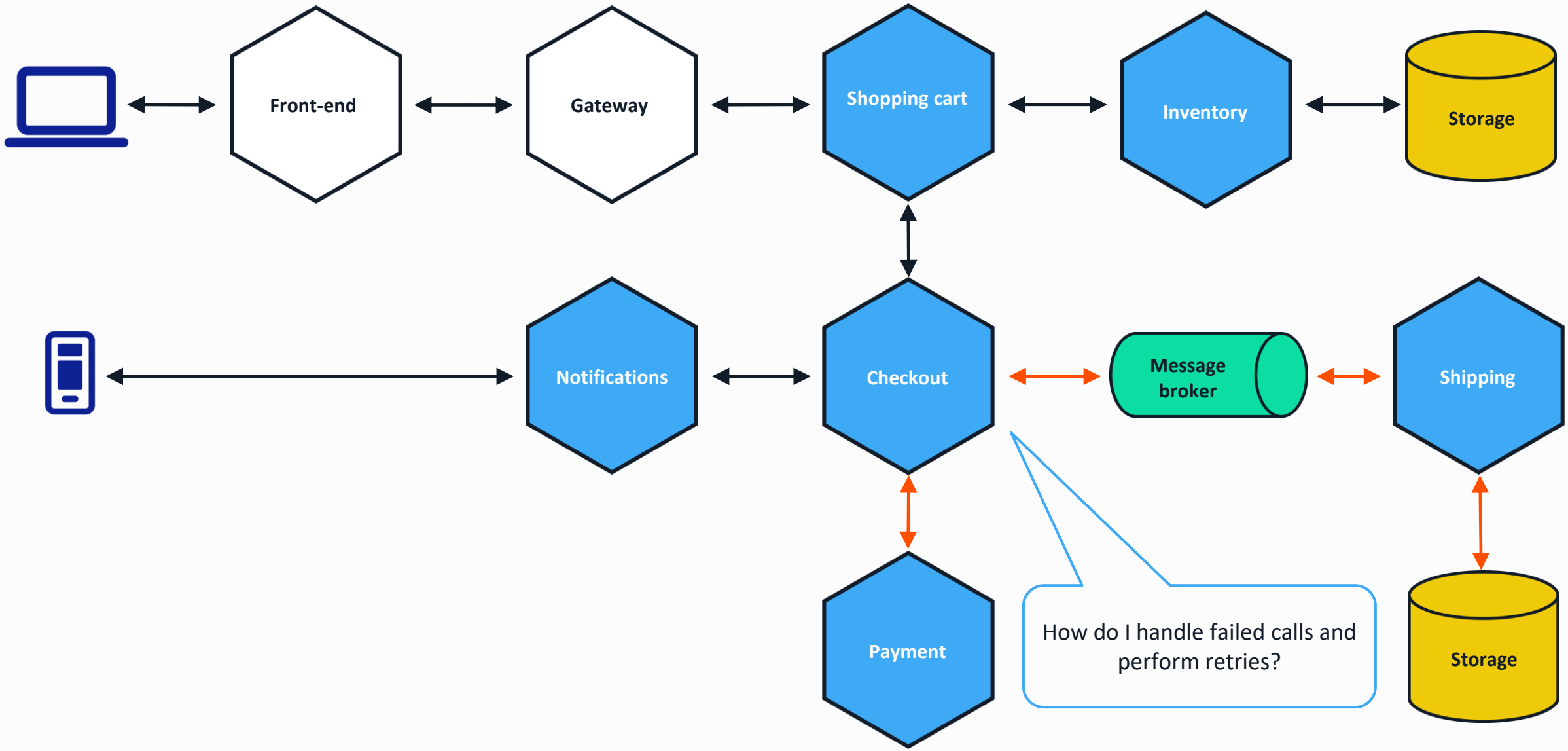
Developer challenges – Distributed tracing



Developer challenges – Access control



Developer challenges – Resiliency

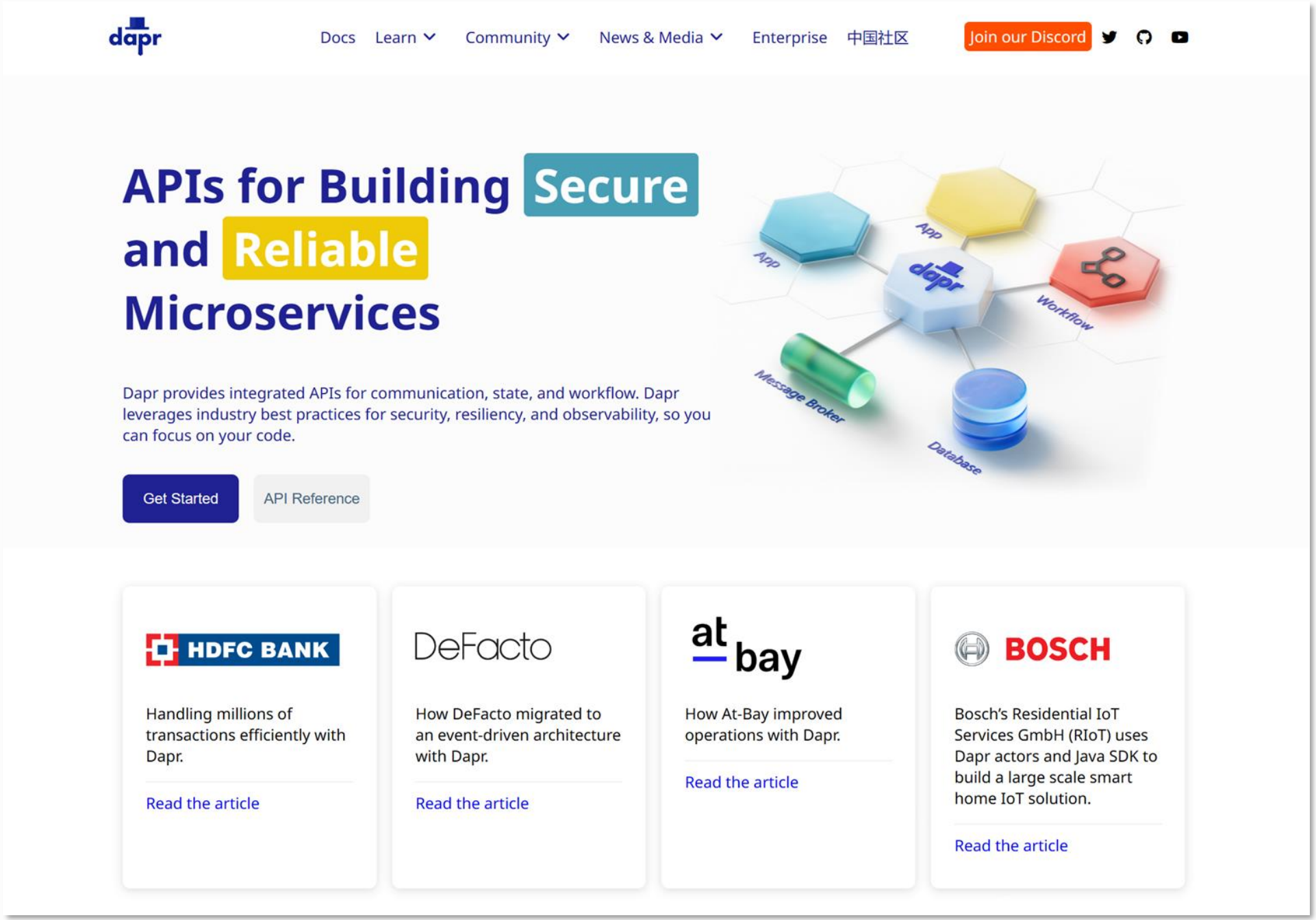


Distributed

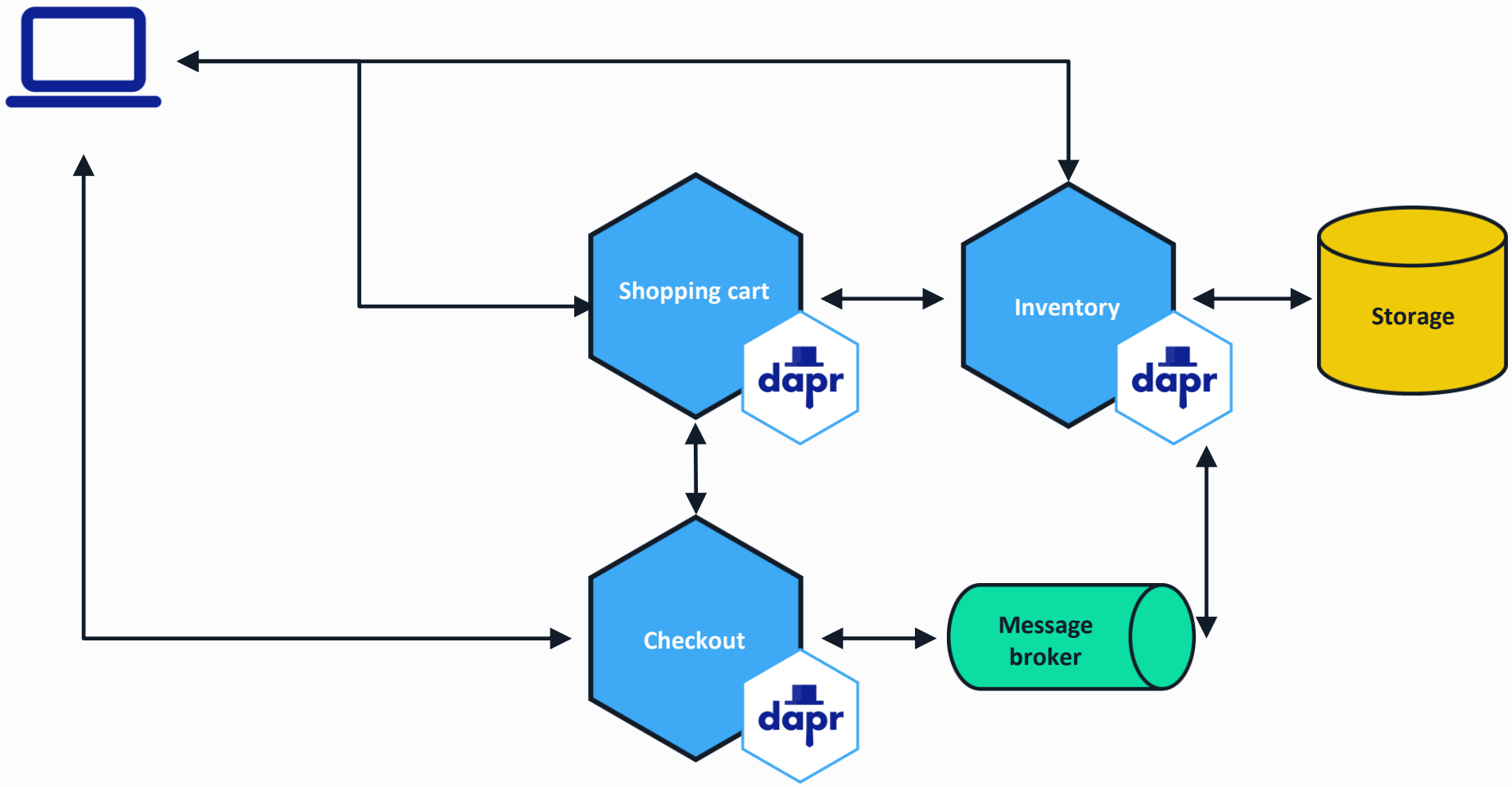
Application

Runtime

dapr.io



Dapr uses a sidecar pattern



The Dapr sidecar provides built-in security, resiliency and observability capabilities.

Speeds up application development by providing an integrated set of APIs for communication, state, and workflow.

State of enterprise developers

Must develop resilient, scalable, distributed apps that interact with services.

Want to focus on writing code, not learning infrastructure.

Trending toward serverless platforms with simple code to cloud pipelines.

Use multiple languages and frameworks during development.

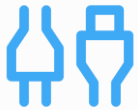
Dapr Goals

Provide an integrated set of APIs

Any language or framework

Includes best practices & standards

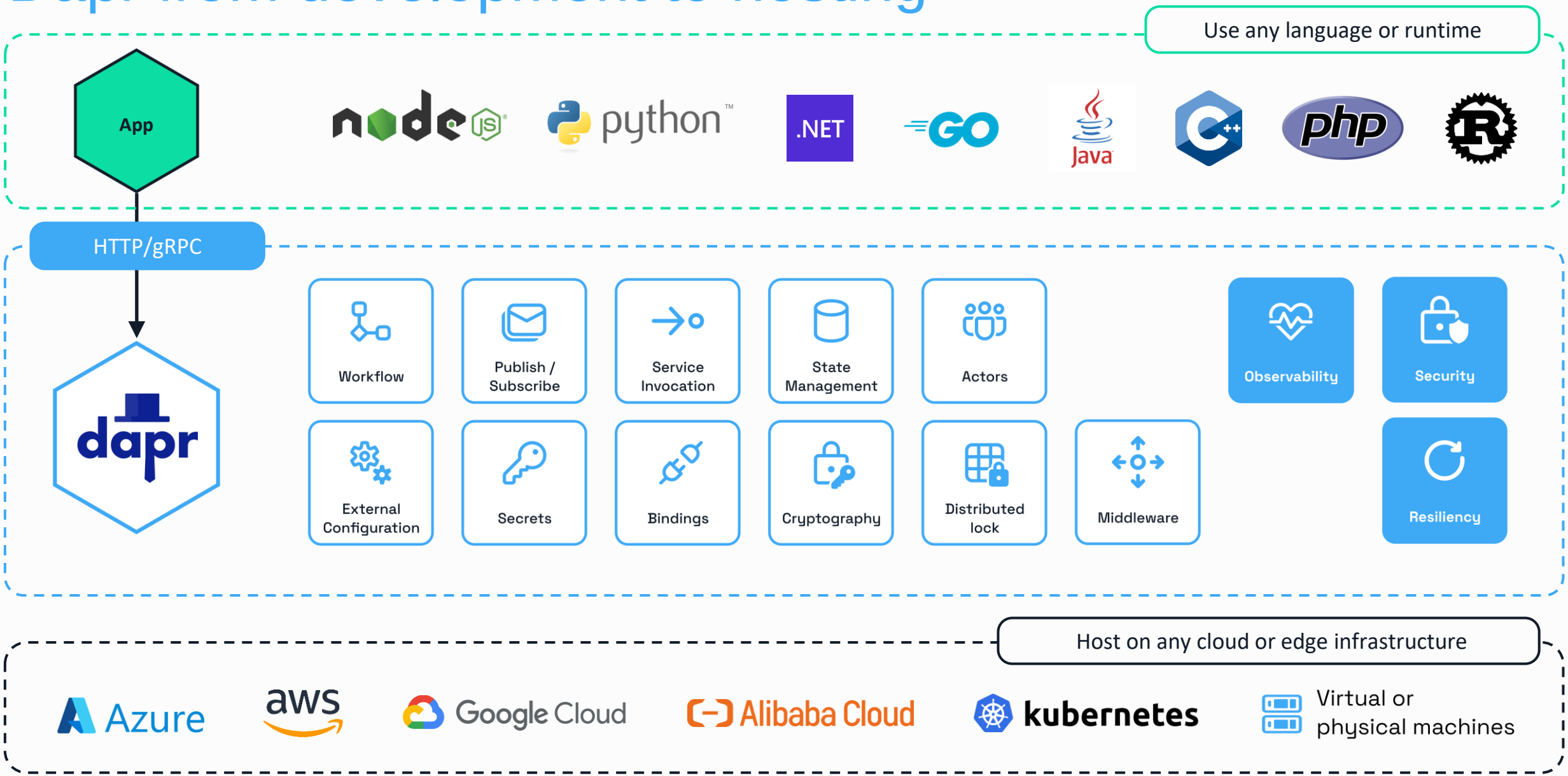
Platform agnostic



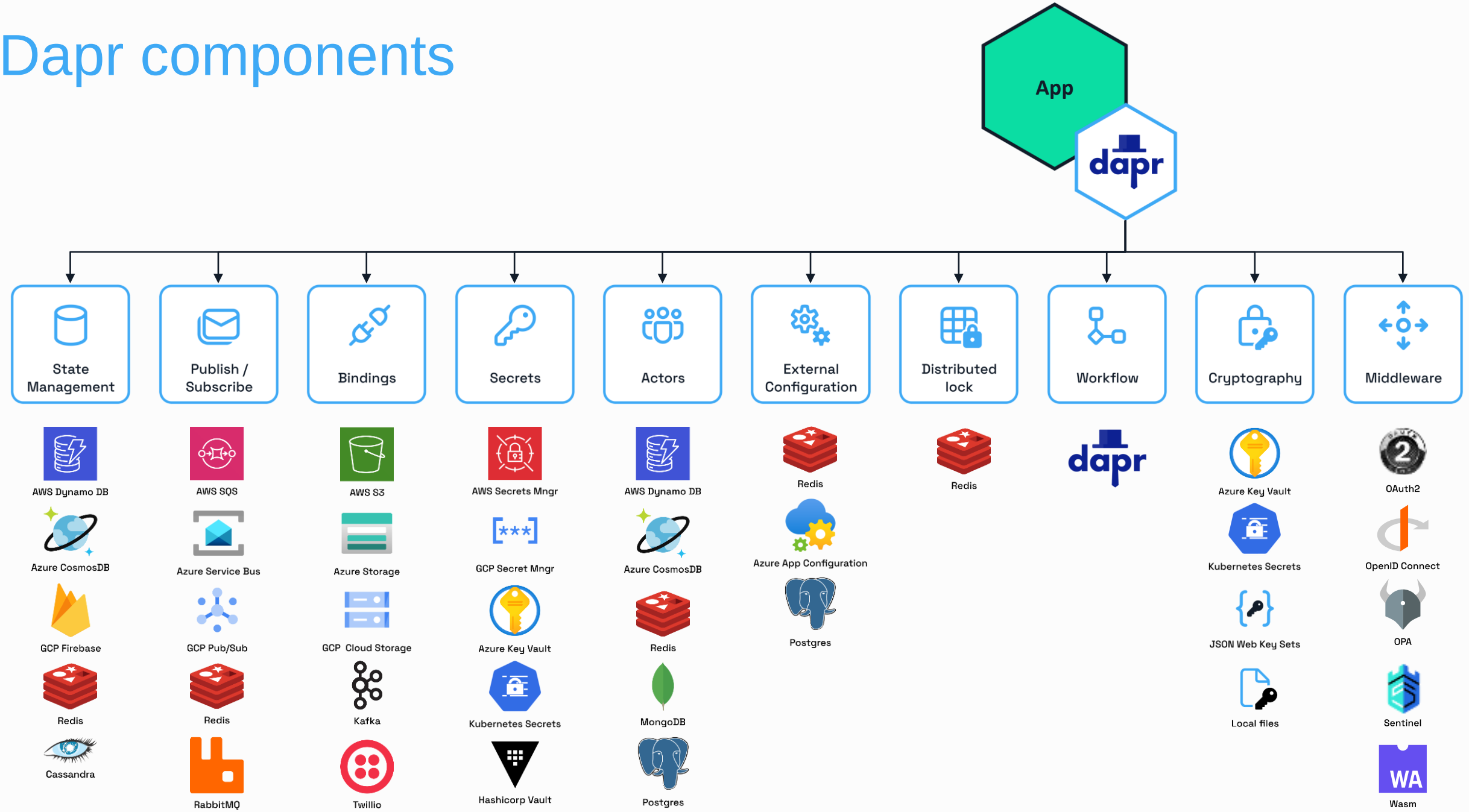
Extensible and pluggable

Community driven, vendor neutral

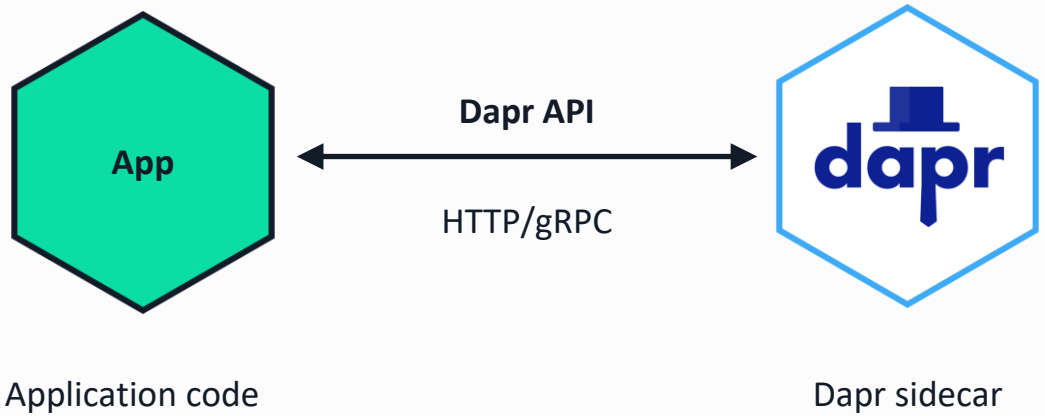
Dapr from development to hosting



Dapr components



Sidecar pattern and the Dapr API



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

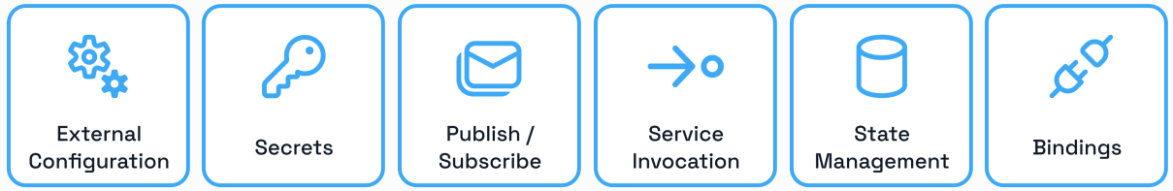
GET http://localhost:3500/v1.0/**state**/inventory/item50

POST http://localhost:3500/v1.0/**publish**/mybroker/order-messages

GET http://localhost:3500/v1.0/**secrets**/vault/dbaccess

POST http://localhost:3500/v1.0-beta1/**workflows**/dapr/businessprocess/start

Using the Dapr APIs



Input Binding Trigger

GET /v1.0/secrets/

App A

POST /v1.0/publish/

Message broker

Subscription

GET /dapr/subscribe/

App C

POST /v1.0/binding/

POST /v1.0/invoke/

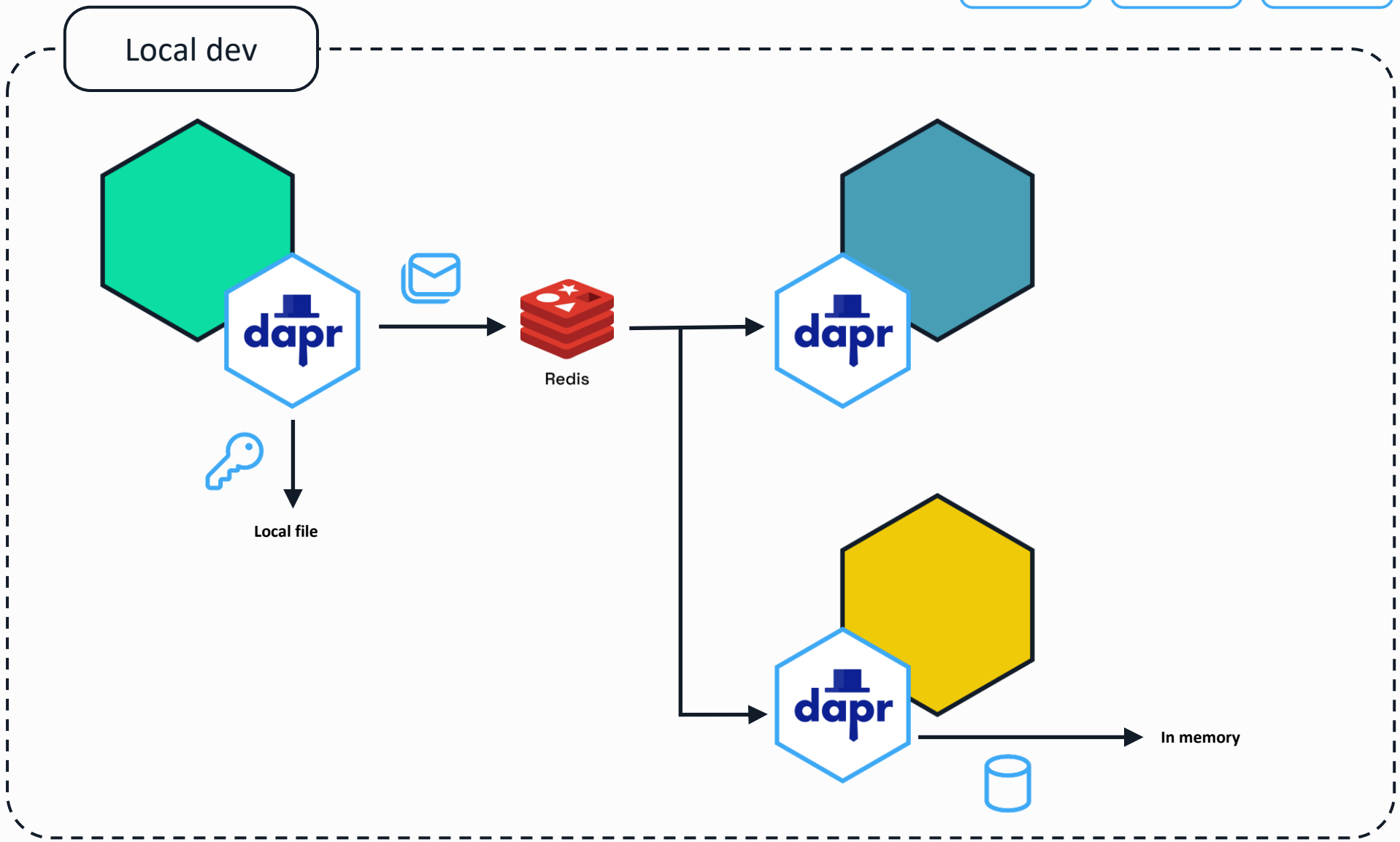
App B

GET /v1.0/configuration/

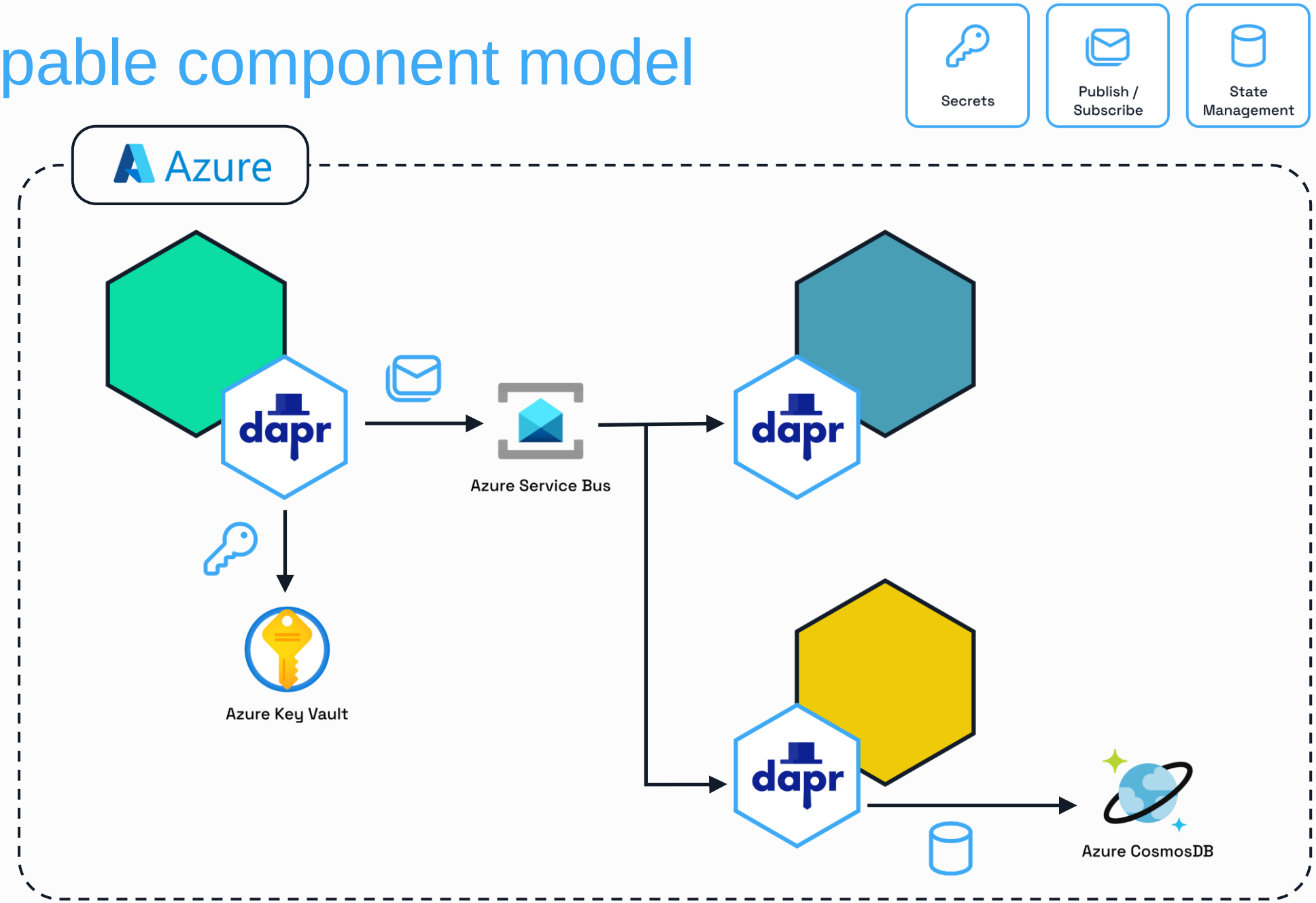
POST /v1.0/state/

Storage

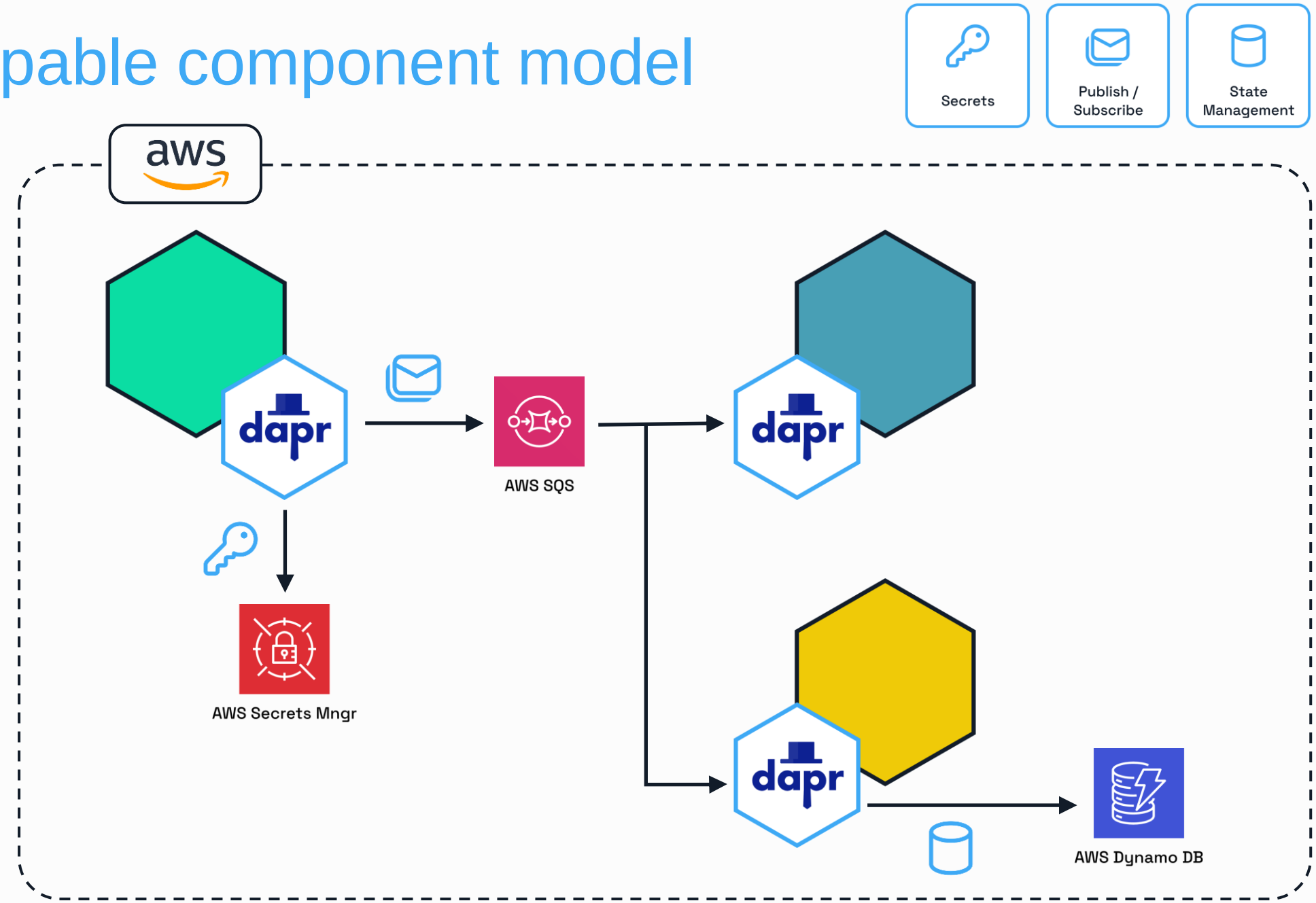
Swappable component model



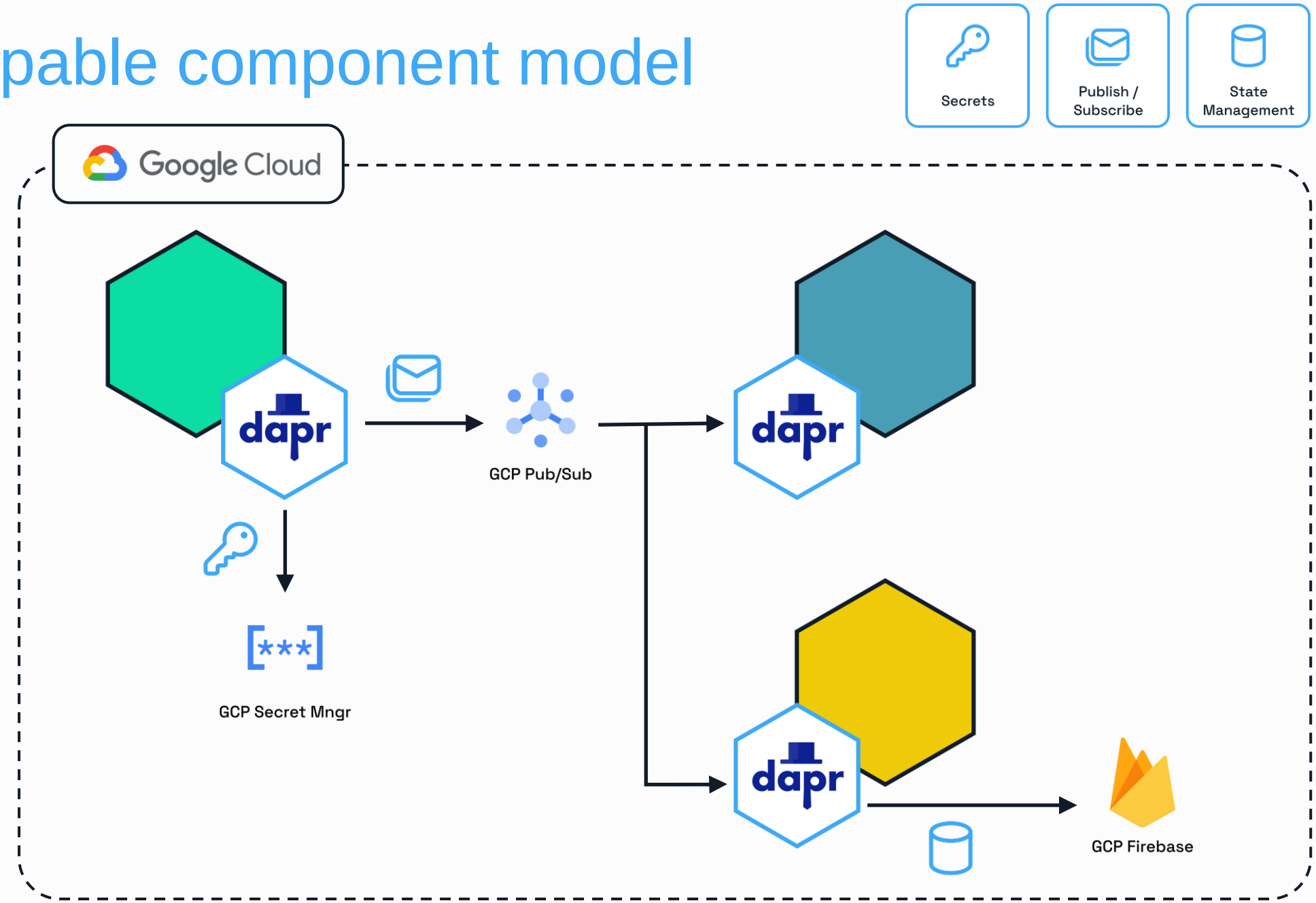
Swappable component model



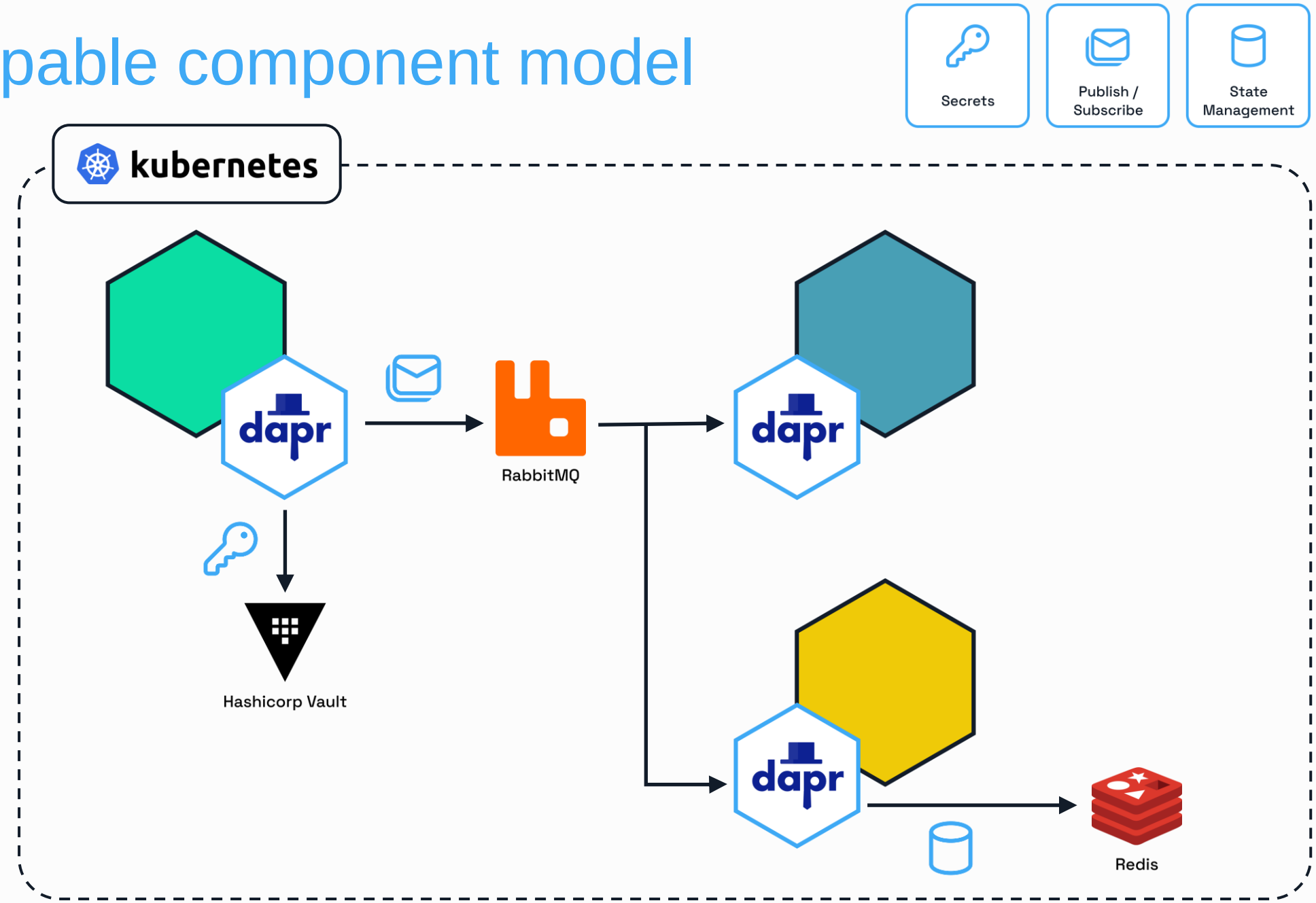
Swappable component model



Swappable component model



Swappable component model

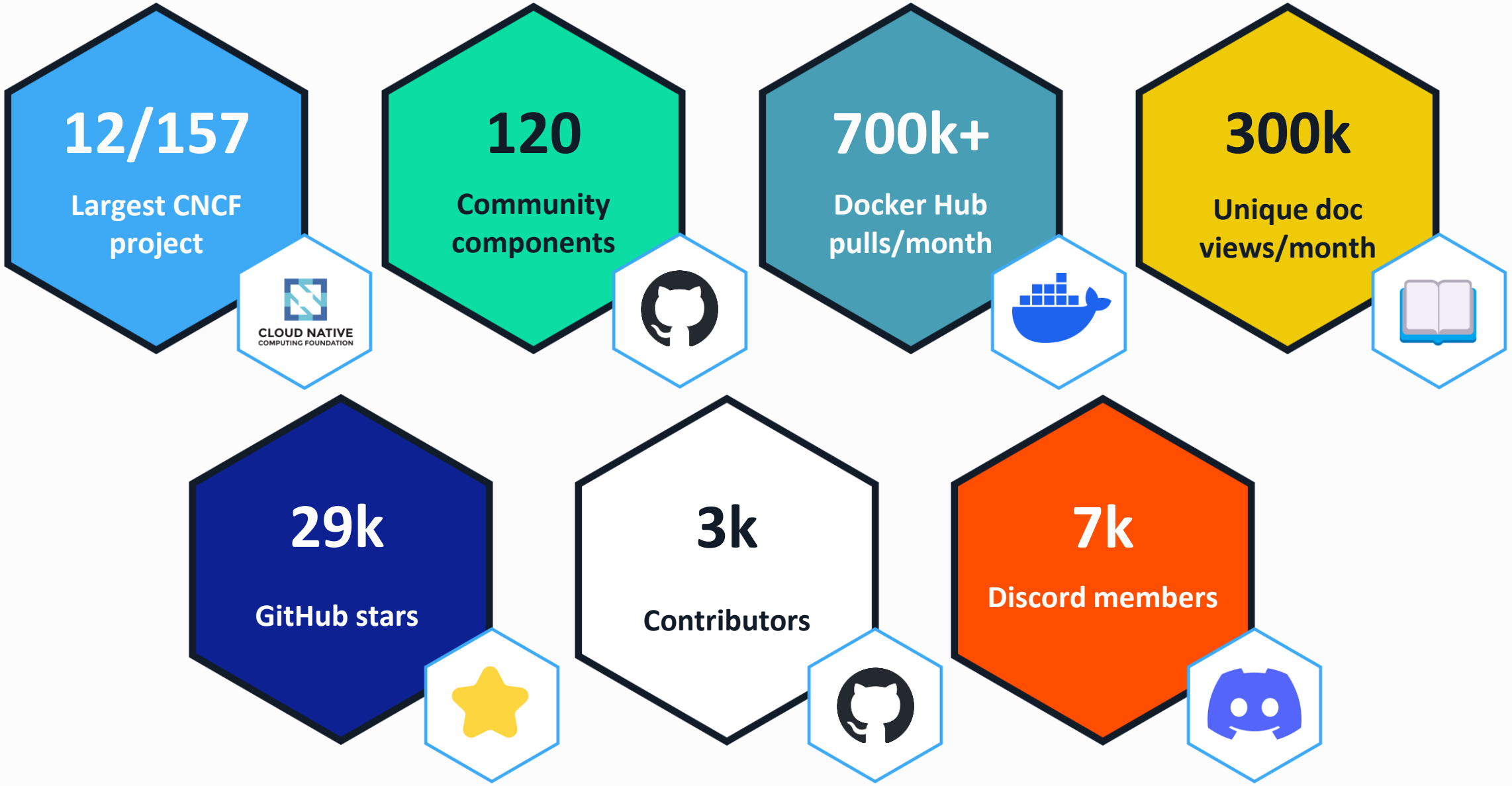


Use Dapr anywhere

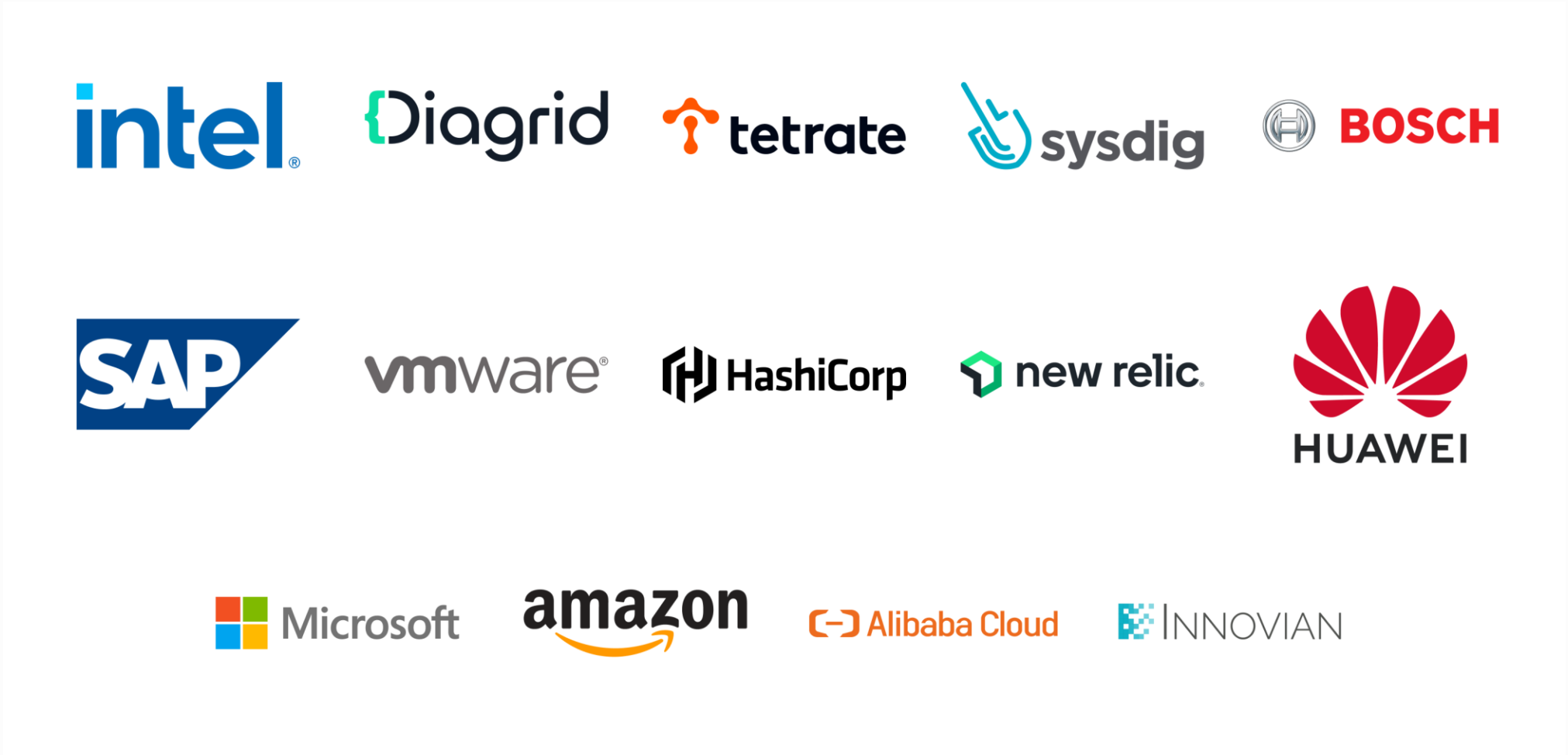
Azure Container Apps

Virtual or
physical machines

Dapr community



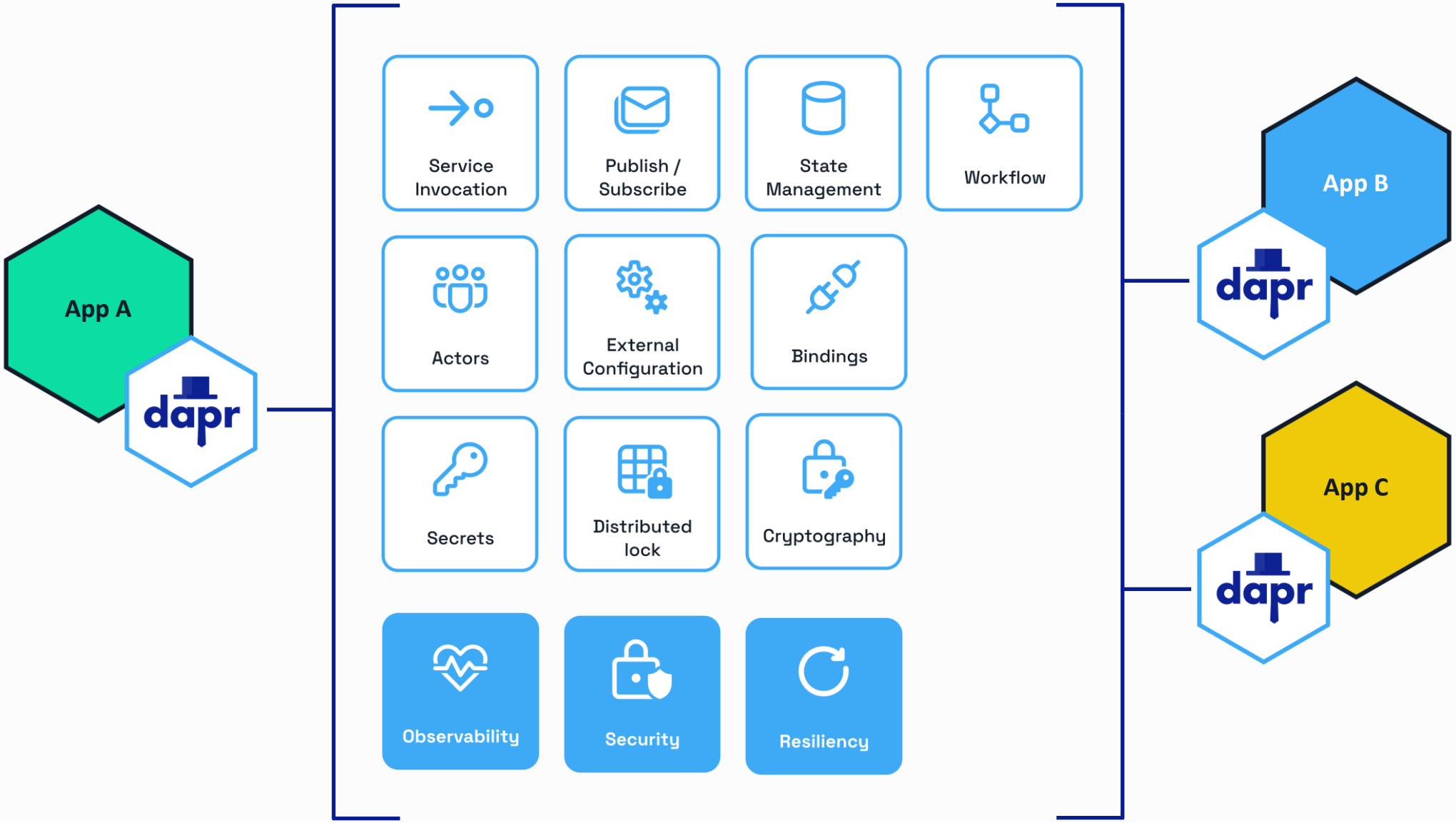
Dapr contributors



Dapr users


IBM
pwc
Microsoft
intel
Alibaba Cloud
ZEISS
NASA
HDFC BANK
CISCO
Rakuten
Lufthansa
FUJITSU
zscaler
BOSCH
DeFacto
VONAGE
at bay
Dotmatics
wortell
IGNITION GROUP

Dapr APIs & Cross cutting concerns



Service Invocation API

https://docs.dapr.io/developing-applications/building-blocks/service-invocation/

Service Invocation

The service invocation API allows synchronous communication between services.

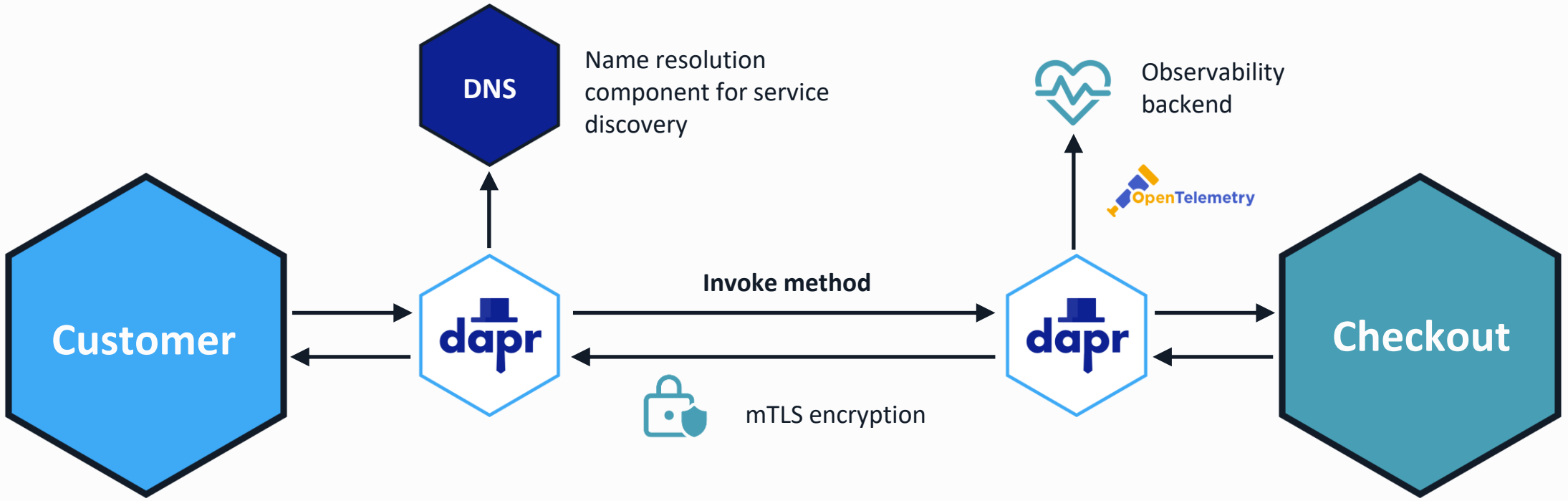
- Service discovery via name resolution components
- Invoke HTTP and gRPC services consistently
- Configurable resiliency policies
- Built-in distributed tracing & metrics
- Access control policies & mTLS
- Chain pluggable middleware components

Service Invocation

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

POST
http://localhost:5100/order

Service Invocation



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

POST
http://localhost:5100/order

Service Invocation Demo

https://docs.dapr.io/getting-started/quickstarts/serviceinvocation-quickstart/

Publish / Subscribe API

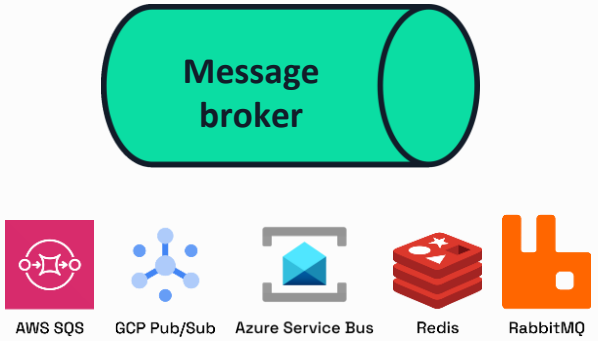
https://docs.dapr.io/developing-applications/building-blocks/pubsub/

Publish / Subscribe

The publish subscribe API allows asynchronous communication between services.

- Integrates with many message brokers and queues
- Guaranteed at least one delivery
- Use declarative or programmatic subscriptions
- Use content-based message routing
- Set dead-letter topics and resiliency policies
- Limit publish and subscribe access by using scopes

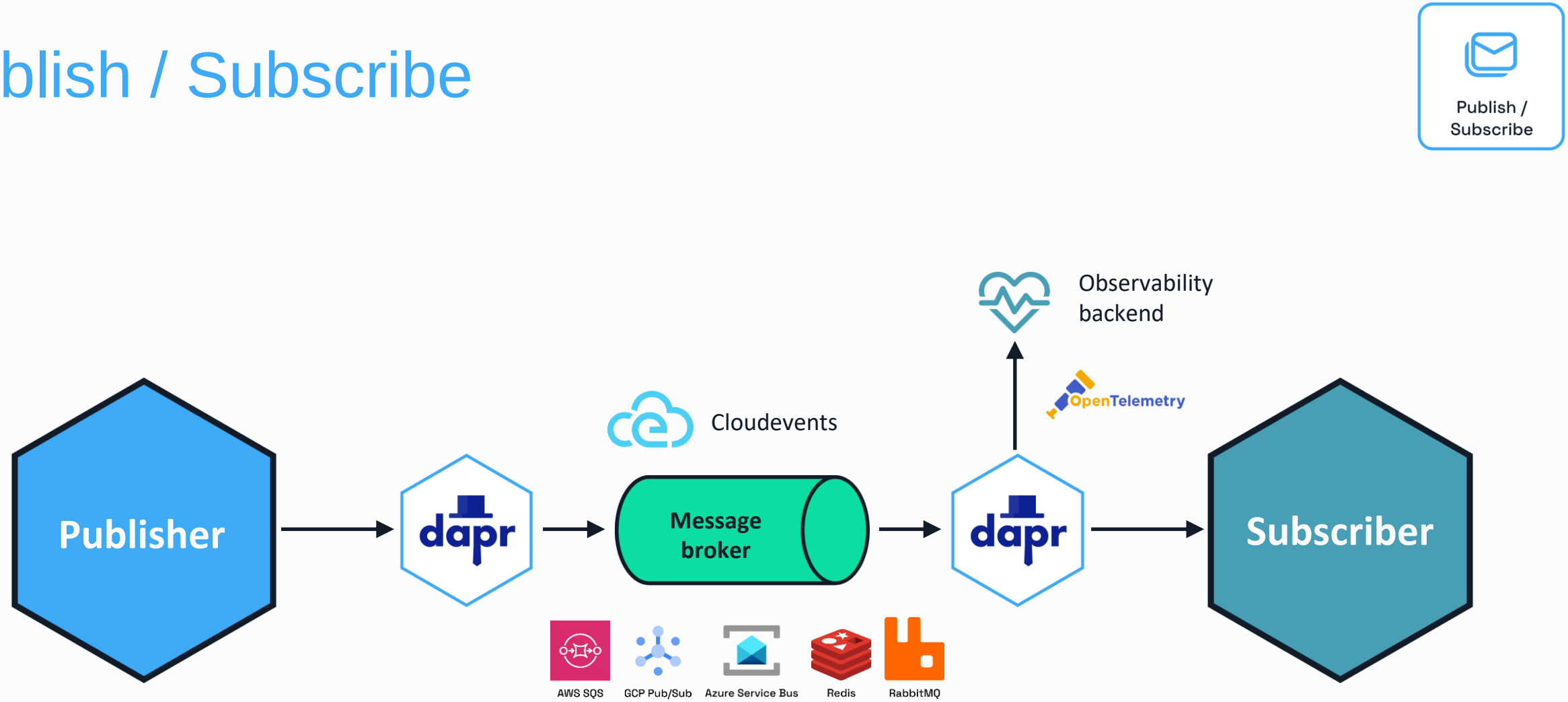
Publish / Subscribe



POST
http://localhost:3500/v1.0/**publish**/mybroker/order-messages

POST
http://localhost:5100/orders

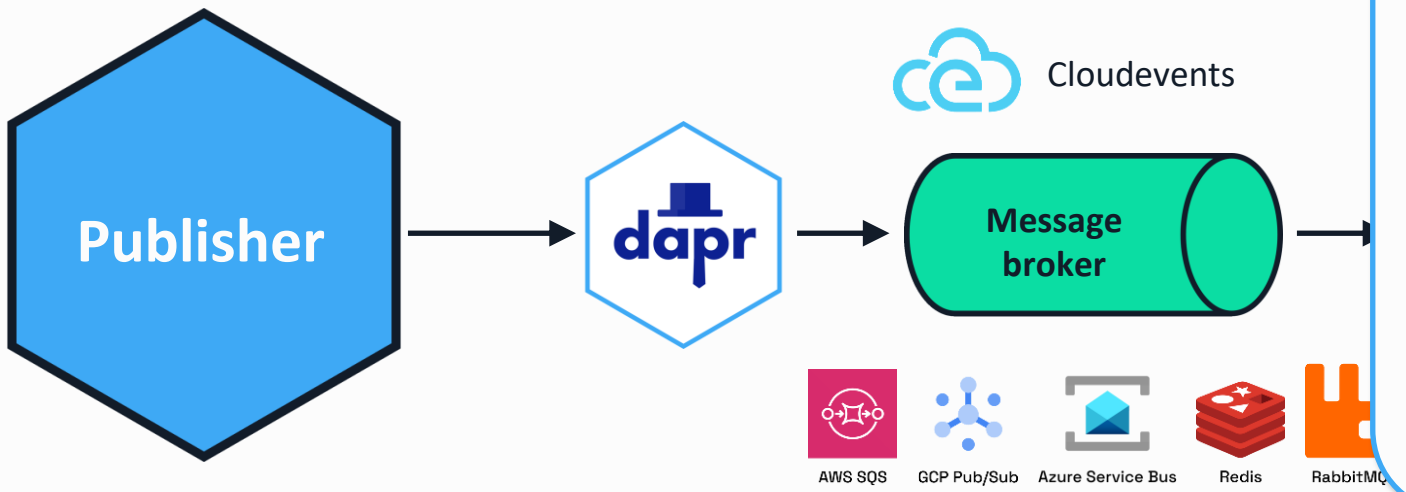
Publish / Subscribe



POST
http://localhost:3500/v1.0/**publish**/mybroker/order-messages

POST
http://localhost:5100/orders

Publish / Subscribe Component




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

POST
http://localhost:3500/v1.0/**publish**/mybroker/order-messages

POST
http://localhost:5100/orders

Publish / Subscribe Demo

https://docs.dapr.io/getting-started/quickstarts/pubsub-quickstart/

State Management API (Key/Value)

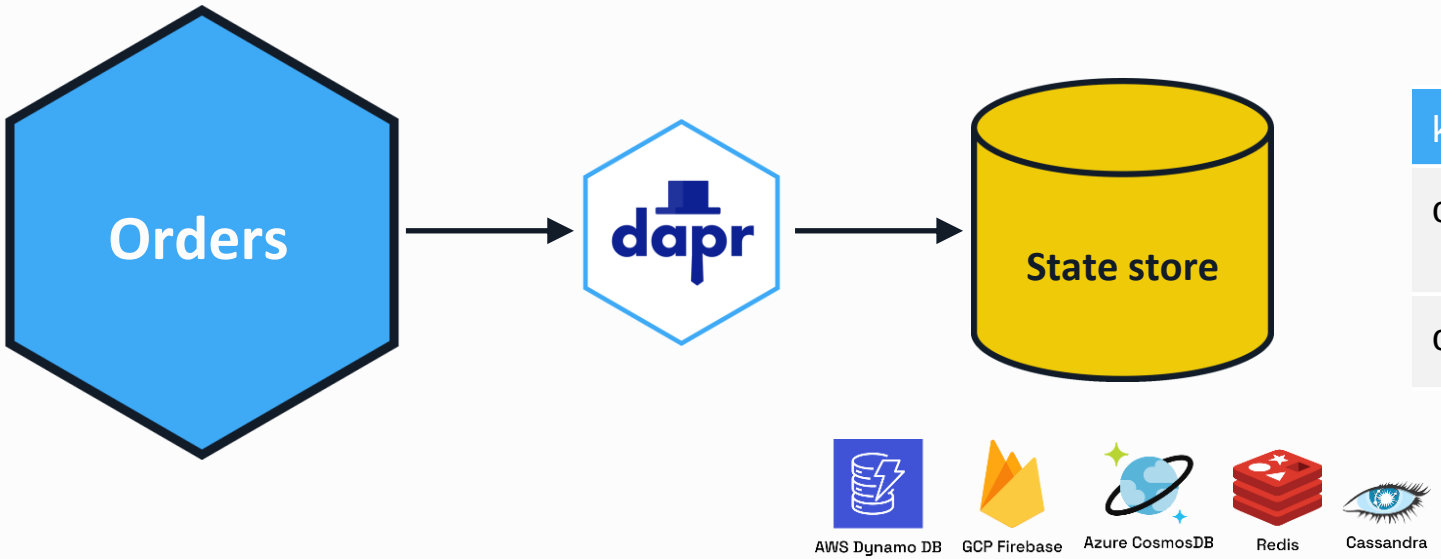
https://docs.dapr.io/developing-applications/building-blocks/state-management/

State Management

The state management API allows key/value pair storage across many supported state stores.

- Integrates with many state stores
- Configurable concurrency and consistency behaviors
- Use bulk operations
- Use resiliency policies
- Limit access by using scopes

State Management (Key/Value)

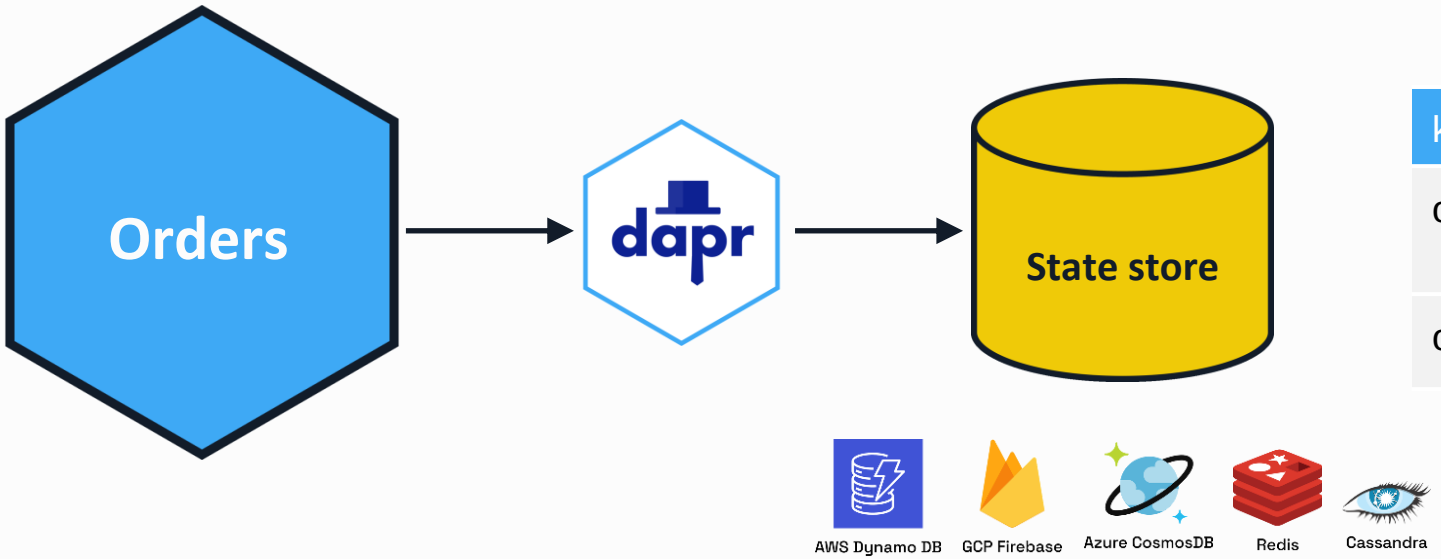


key	field	value
orders||order1	data	"{orderId:1}"
orders||order1	version	1

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

```
[{
  “key”: “order1”,
  “value”: “{orderId: 1}”
}]
```

State Management (Key/Value)



key	field	value
orders||order1	data	"{orderId:1}"
orders||order1	version	1

GET

http://localhost:3500/v1.0/state/mystatestore/order1

State Management (Key/Value) Demo

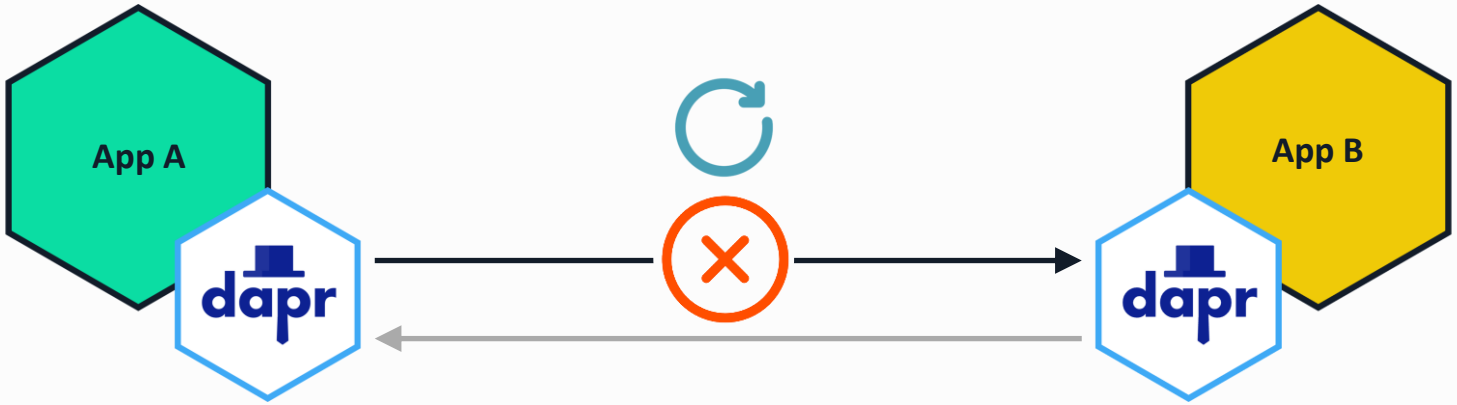
https://docs.dapr.io/getting-started/quickstarts/statemanagement-quickstart/

Resiliency

https://docs.dapr.io/concepts/resiliency-concept/

Service invocation resiliency

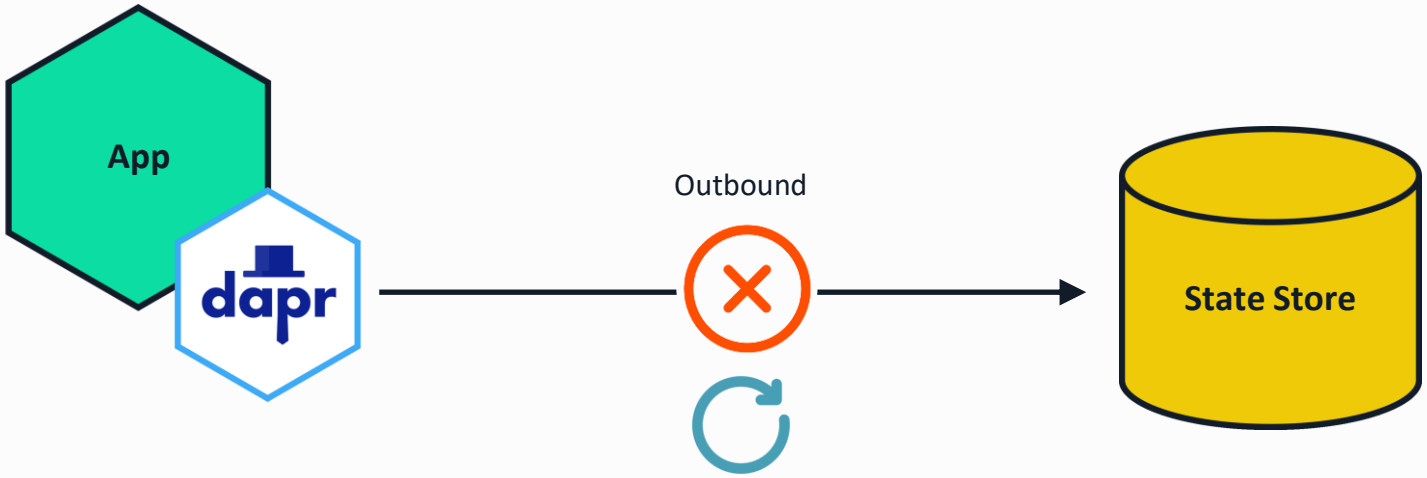
The built-in service invocation retries are always performed with a backoff interval of 1 second up to a threshold of 3 times.



Additionally, service invocation resiliency polices for *retries*, *timeouts* and *circuit breakers* can be applied.

Outbound component resiliency

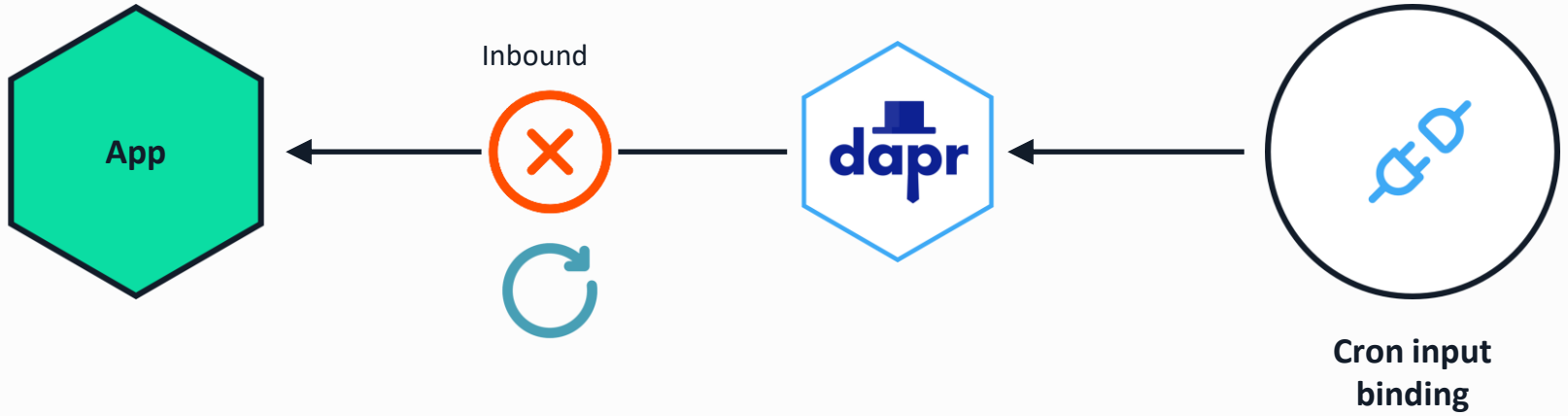
Component resiliency polices can be applied to outbound component calls. For example, calls to a state store.



Additionally, some components have retry capabilities built-in. The policies are configured on a per component basis.

Inbound component resiliency

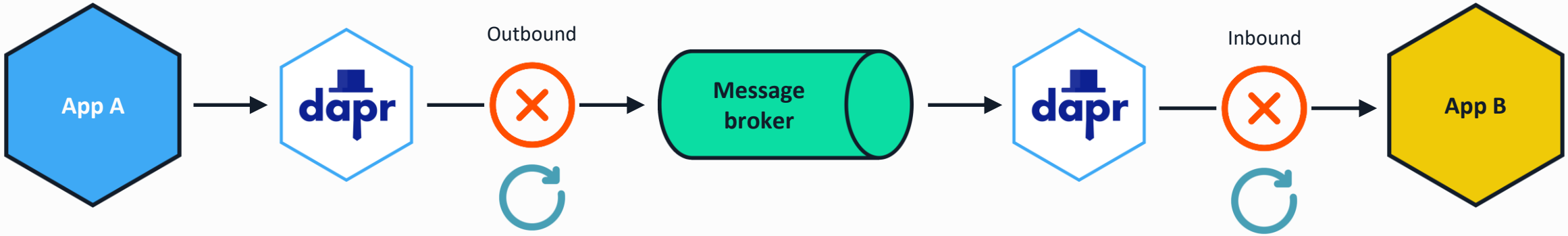
Resiliency polices can be applied to inbound component calls. For example, pub/sub subscriptions and input bindings.



Pub/Sub resiliency

Outbound component resiliency policies for can be applied to message publishing.

Inbound component resiliency polices can be applied to subscriptions when delivering messages.



Additionally, many pub/sub components have *retry* capabilities built-in. The policies are configured on a per component basis.

Resiliency

Resiliency patterns can be applied across Dapr APIs:

- Retries
- Timeouts
- Circuit breakers

Declarative and decoupled from application code.

Available across all component types, service invocation, and actors.

```yaml
apiVersion: dapr.io/v1alpha1
kind: Resiliency
metadata:
  name: myresiliency
scopes:
  - order-processor

spec:
  policies:
    retries:
      retryForever:
        policy: constant
        duration: 5s
        maxRetries: -1

    circuitBreakers:
      simpleCB:
        maxRequests: 1
        timeout: 5s
        trip: consecutiveFailures >= 5

  targets:
    components:
      statestore:
        outbound:
          retry: retryForever
          circuitBreaker: simpleCB
```

Hosting modes

Hosting modes

Self-hosted

Run **`dapr init`** to install Docker images.

Run any app with a Dapr side car using dapr run.

Virtual/Physical machines

Self-deploy Dapr control plane and Hashicorp Consul per machine.

Use the Dapr Installer Bundle for airgapped environments.

Run any app with a Dapr side car using **`dapr run`**.

Kubernetes

Run **`dapr init -k`** to install Dapr (or use Helm). Integrated Dapr control plane.

Deploys placement, operator, sentry and injector pods.

Automatically injects a Dapr sidecar into all annotated pods.

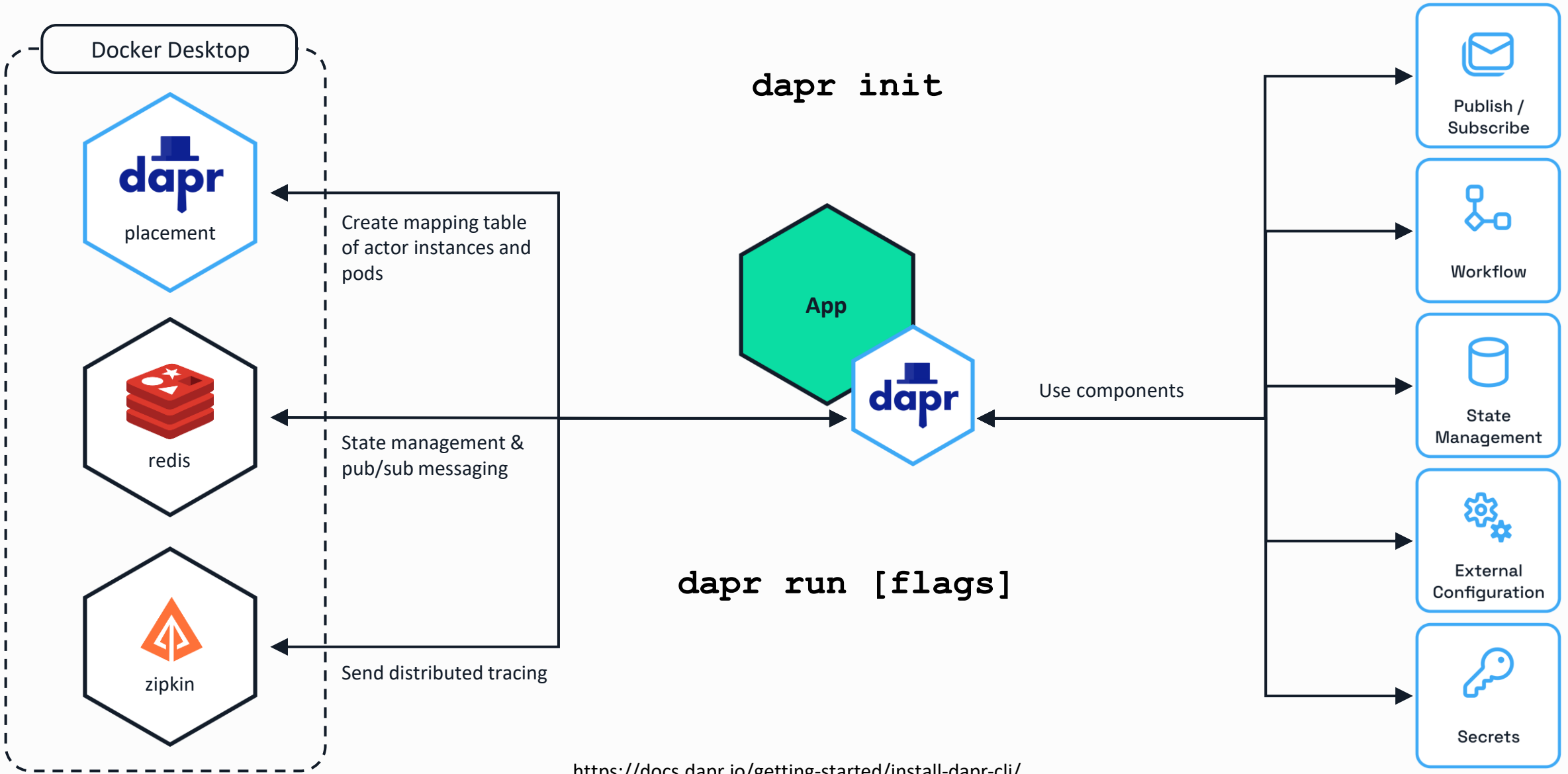
Hosting modes

Serverless

The Dapr side car is hosted by a provider.

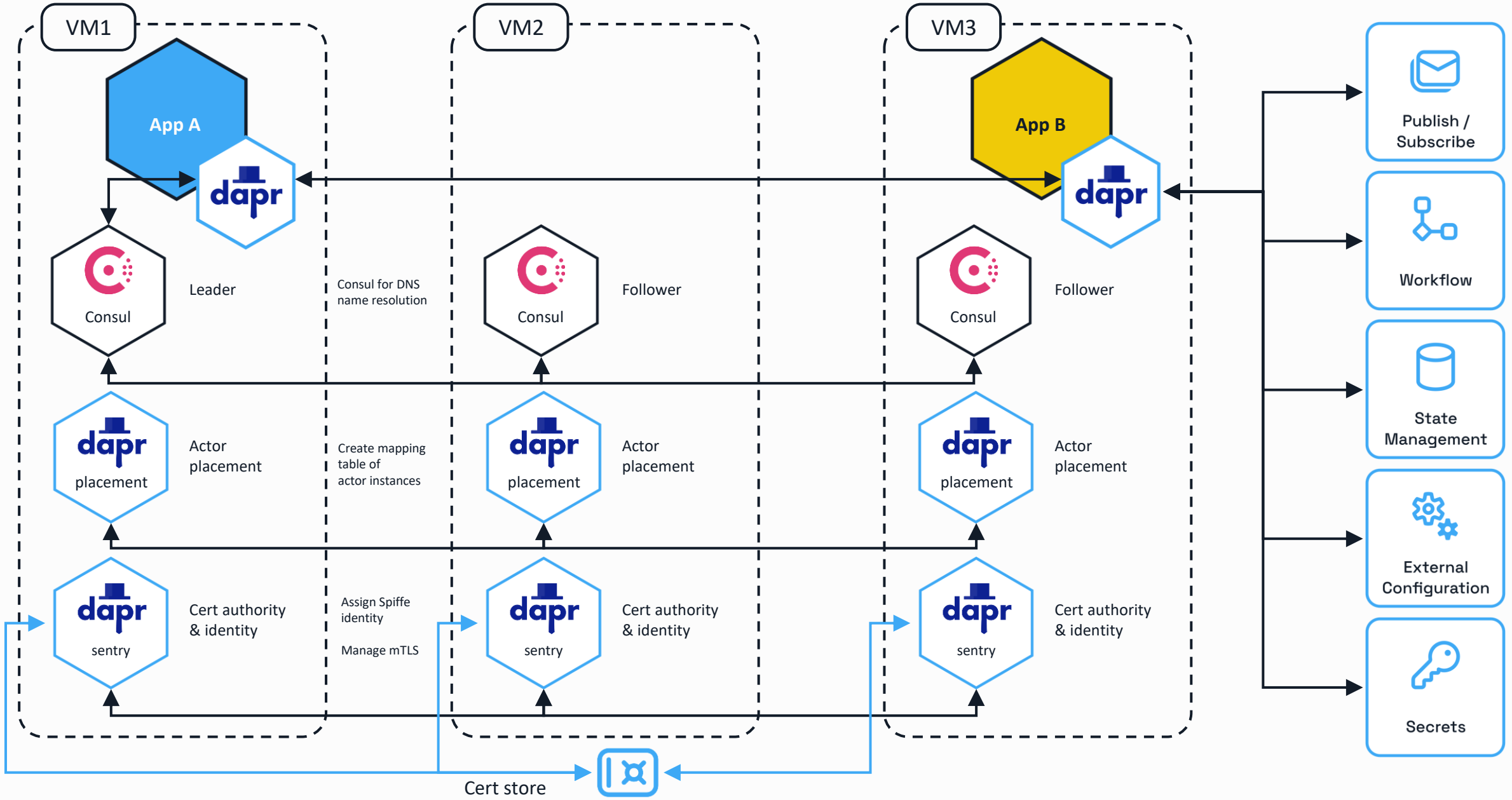
You only manage your applications.

Local development with the Dapr CLI

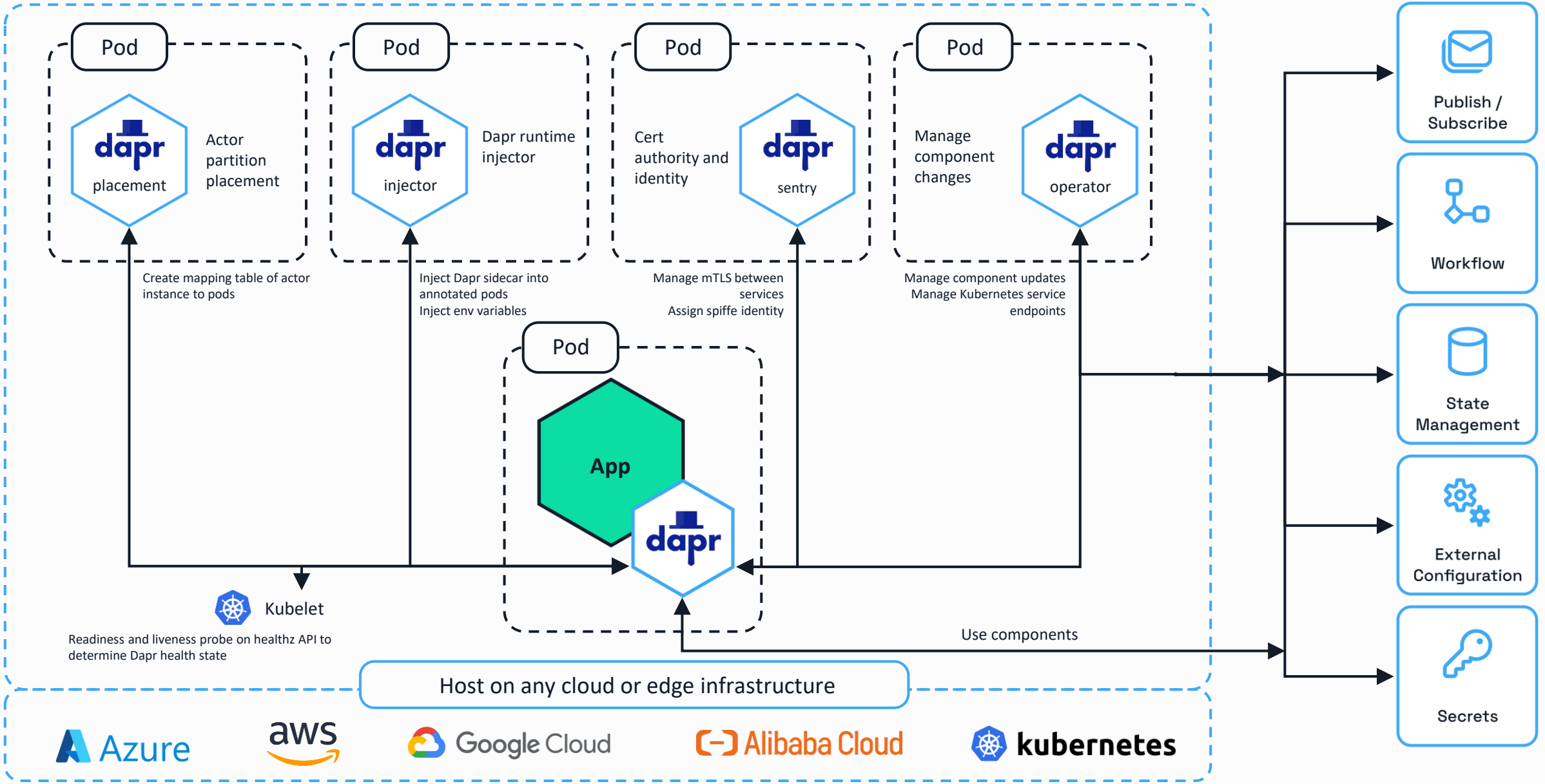


https://docs.dapr.io/getting-started/install-dapr-cli/

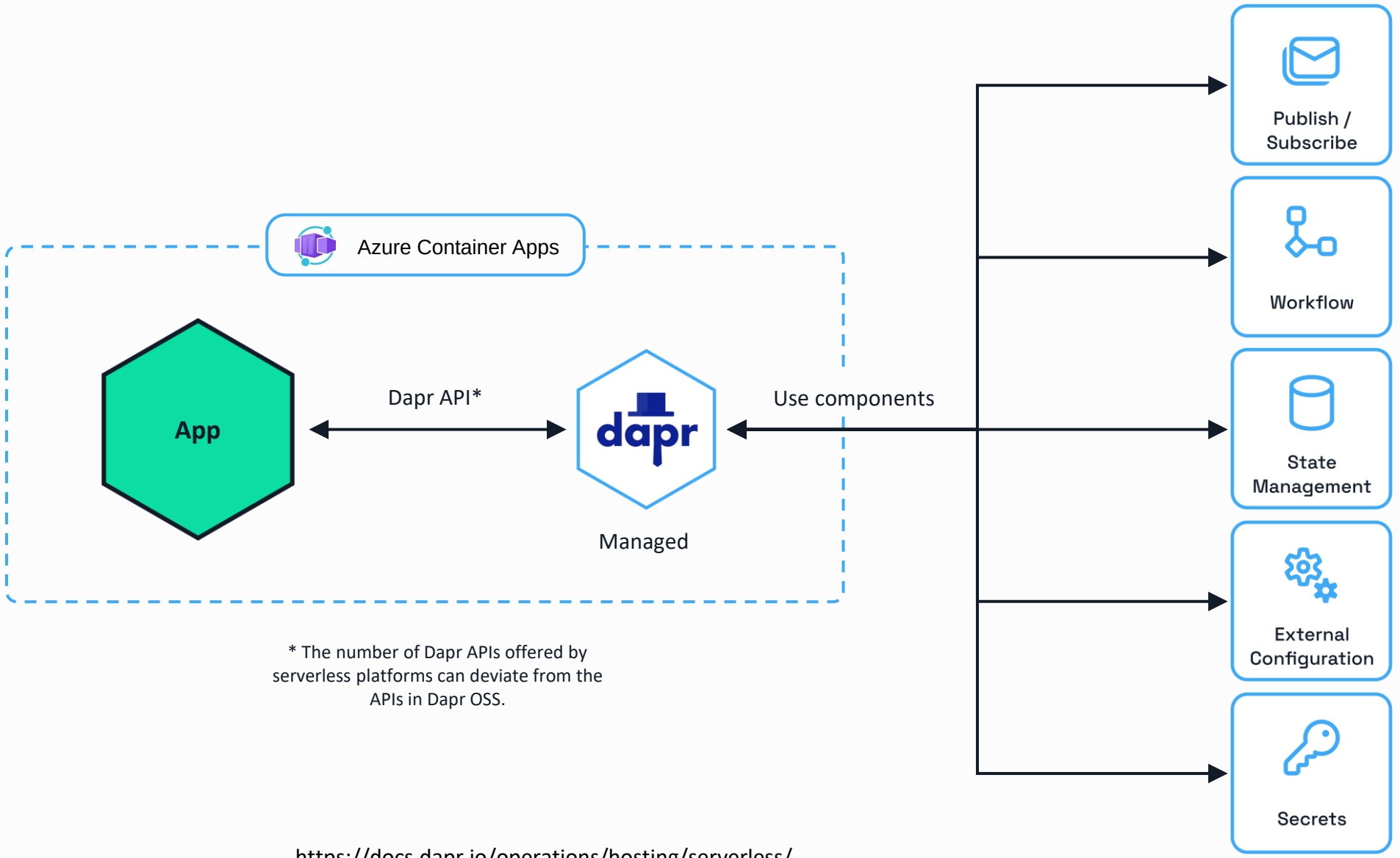
Dapr in self-hosted mode on VMs



Dapr on Kubernetes



Serverless



* The number of Dapr APIs offered by serverless platforms can deviate from the APIs in Dapr OSS.

https://docs.dapr.io/operations/hosting/serverless/

Dapr Resources

dapr.io

bit.ly/dapr-youtube

bit.ly/dapr-quickstarts

bit.ly/dapr-discord

@daprdev

dapr.io

Claim the Dapr Community Supporter badge!

bit.ly/dapr-supporter