# Simplificando o desenvolvimento de microsserviços com o Dapr

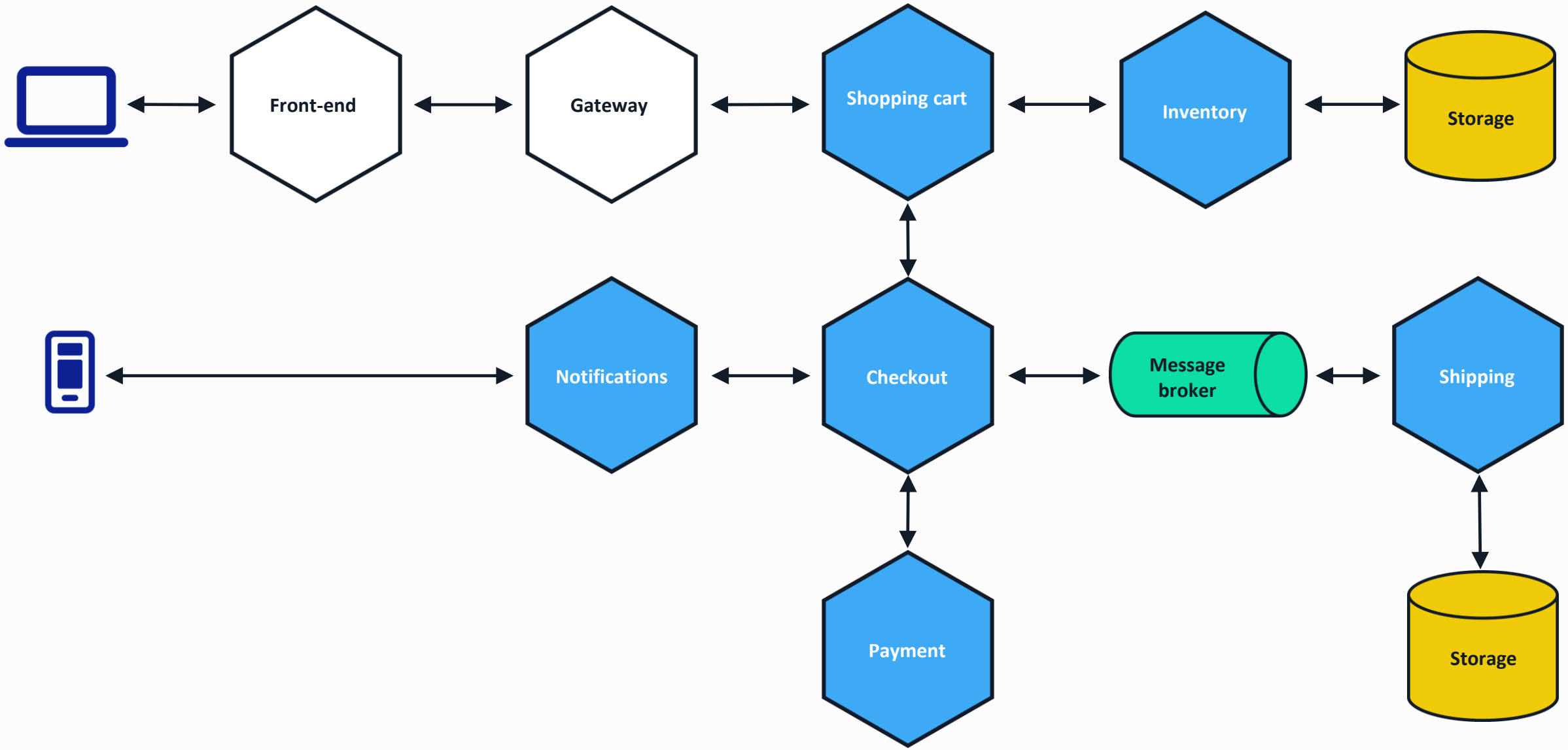

Walter Silvestre Coan

# Walter Silvestre Coan

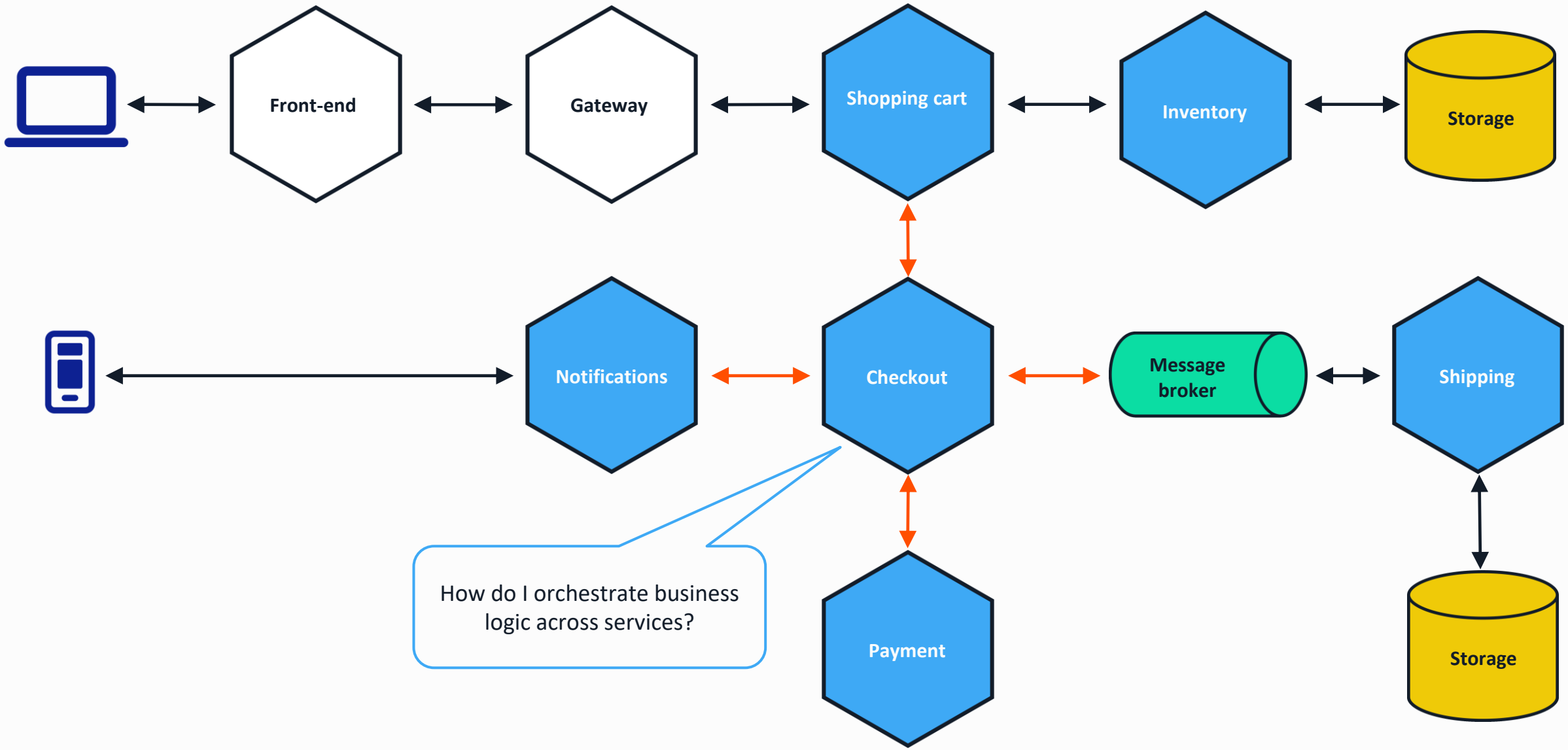
**www.linkedin.com/in/waltercoan/**

- Microsoft MVP na categoria Internet das Coisas
- Dapr Meteors 2025
- Mestre em Sistemas Distribuídos e Redes de sensores sem fio PUCPR
- Instrutor autorizado Microsoft, AWS, NVIDIA na Ka Solution
- Professor na UNIVILLE

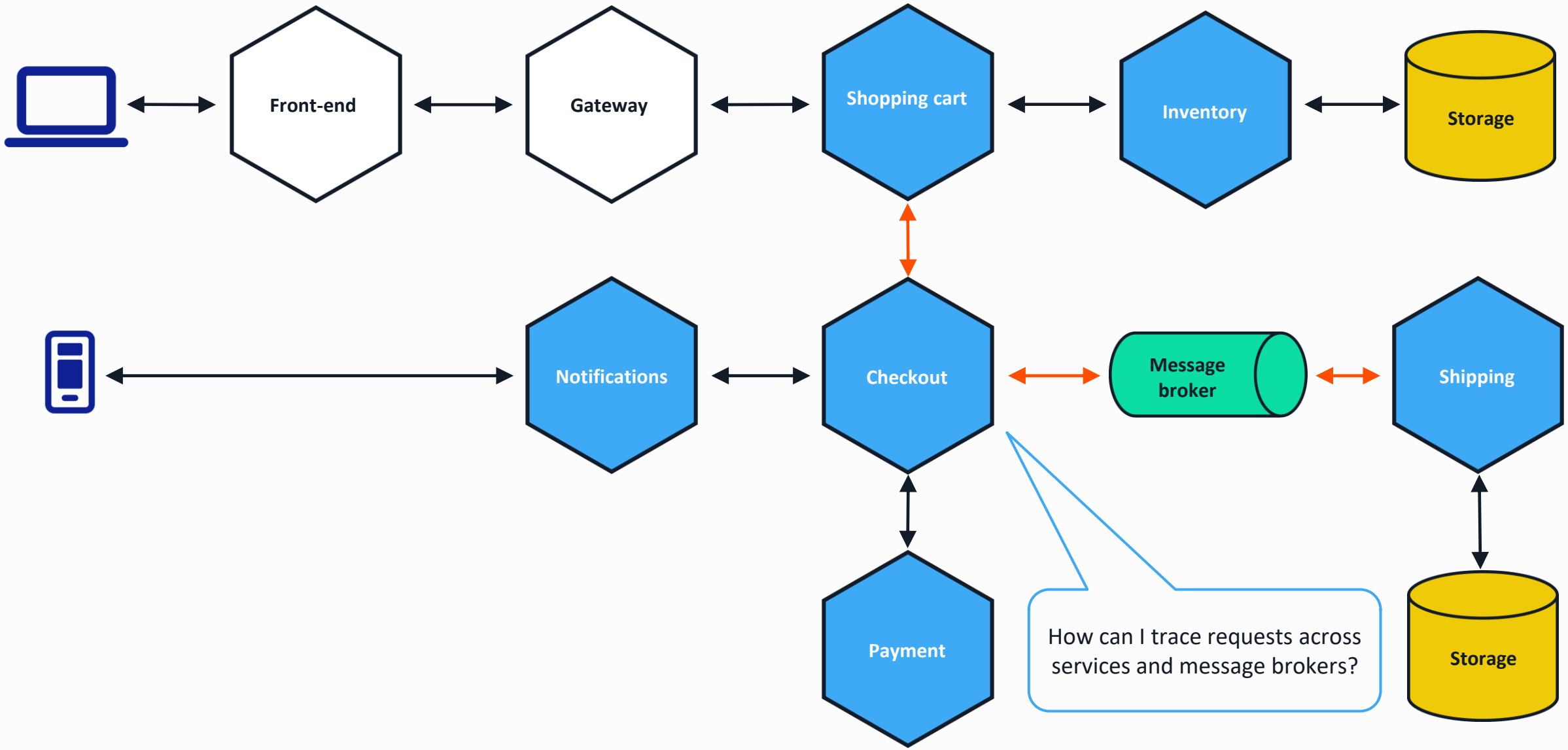

# Distributed applications



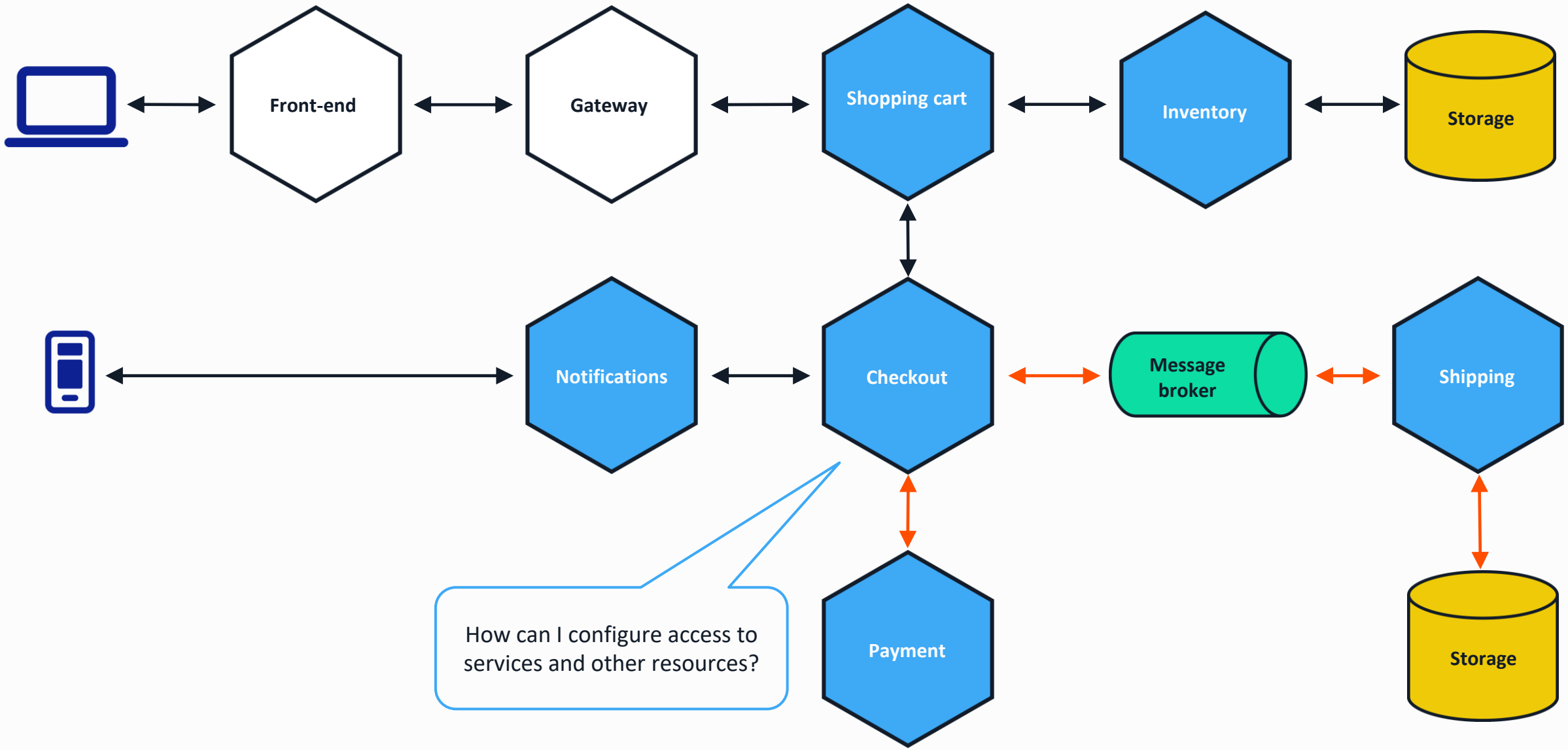
# Developer challenges – Service orchestration



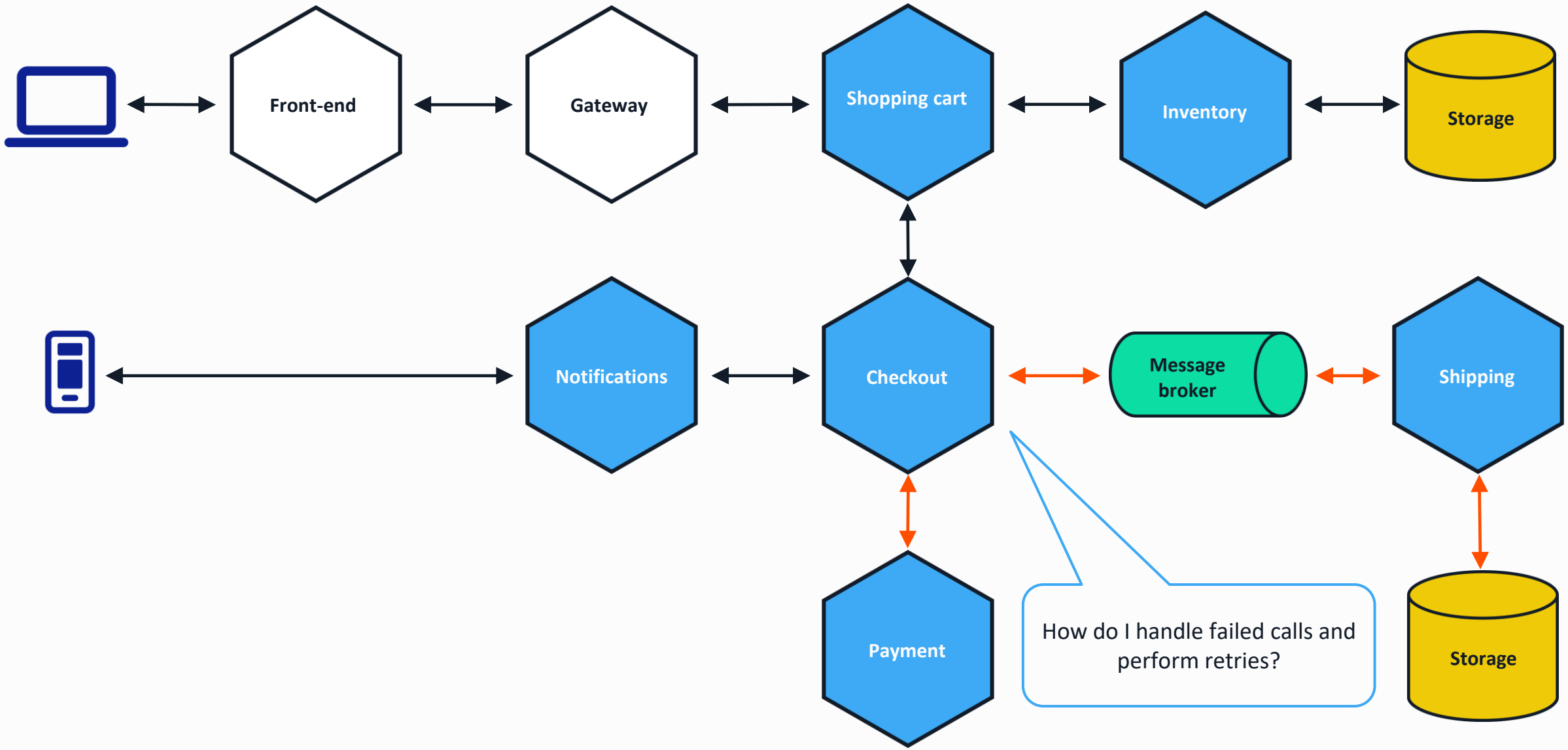
# Developer challenges – Distributed tracing



# Developer challenges – Access control



# Developer challenges – Resiliency



# Distributed

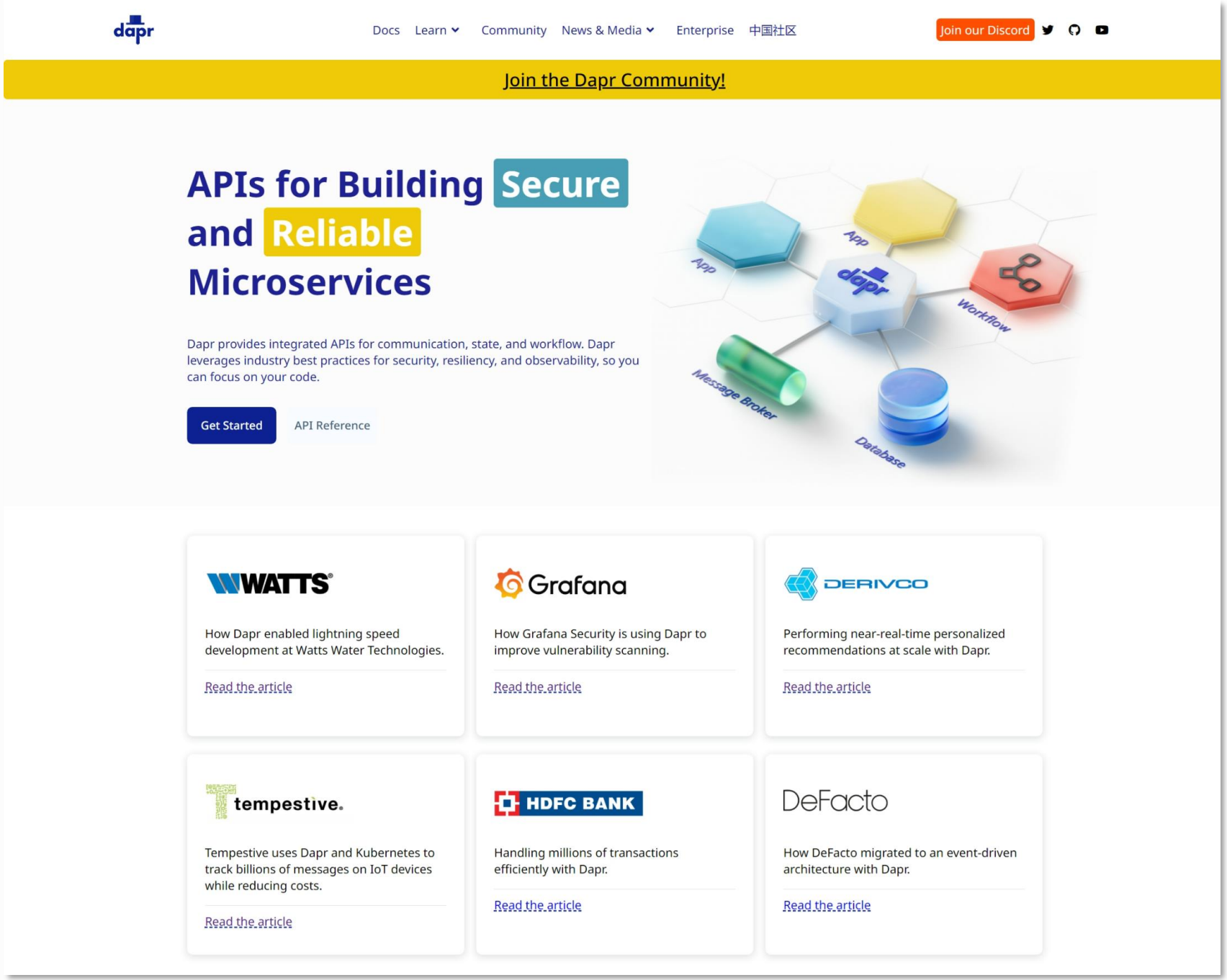
# Application

# Runtime

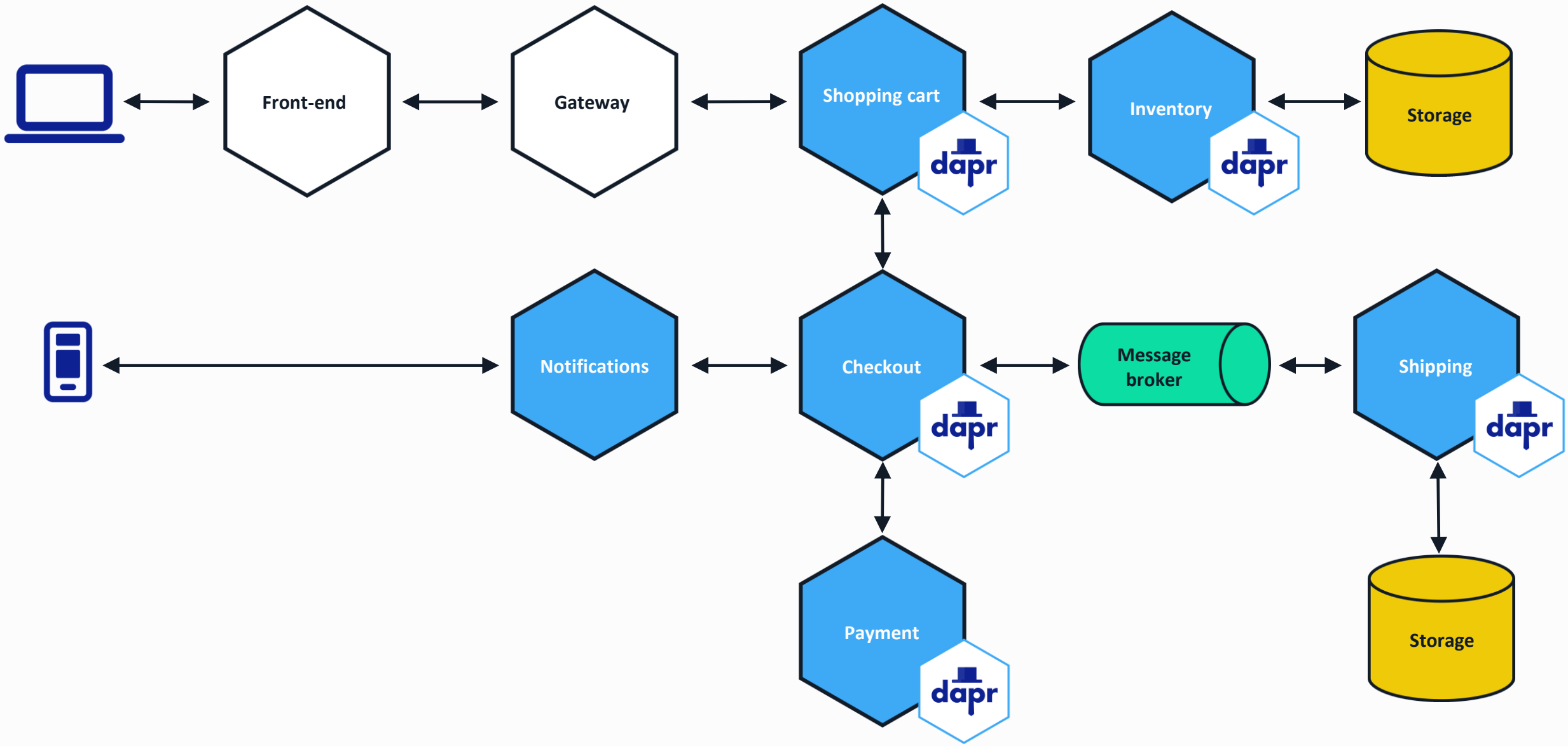
dapr.io

CLOUD NATIVE
COMPUTING FOUNDATION

Graduated project



# Dapr uses a sidecar pattern



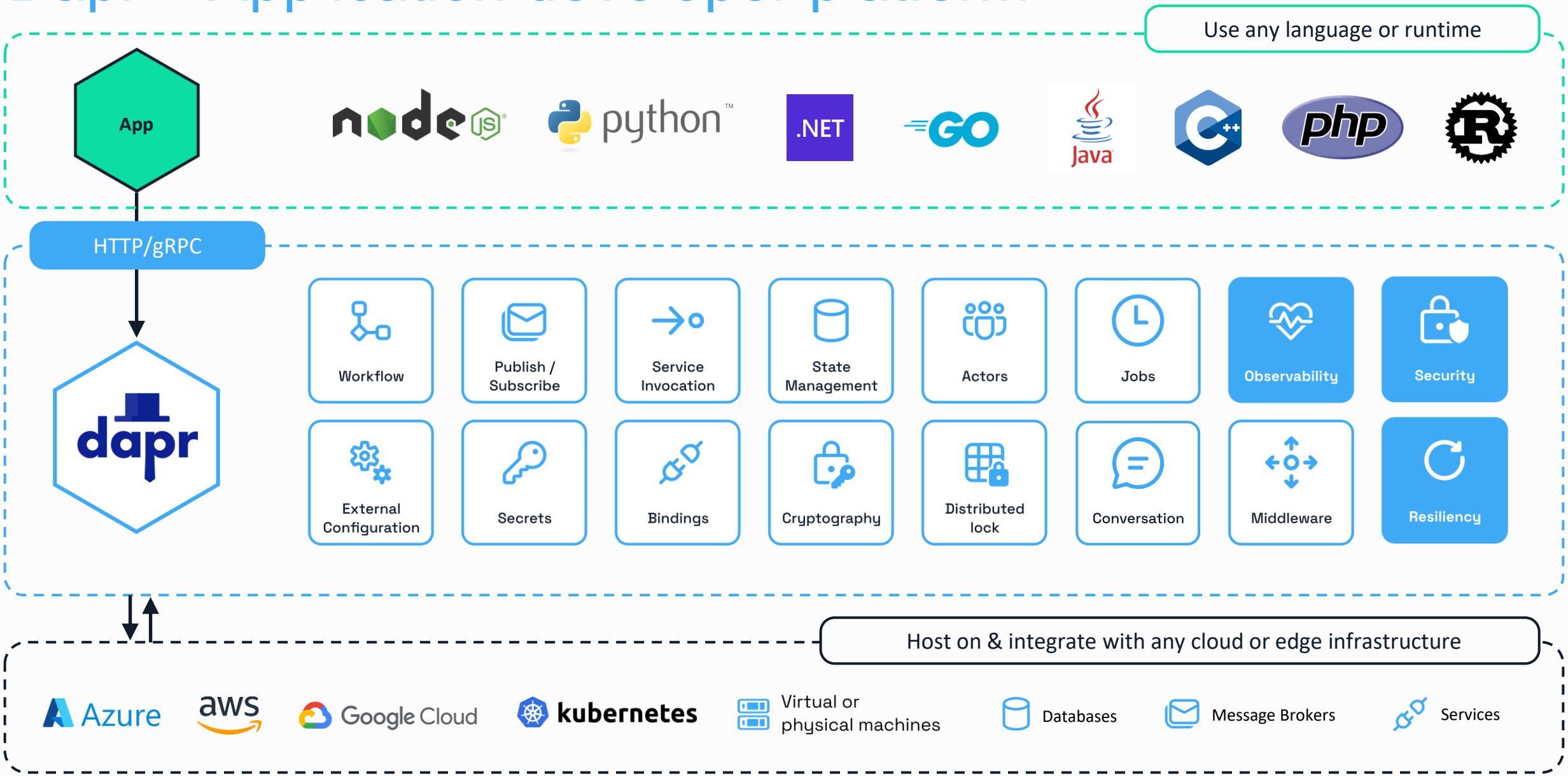
The Dapr sidecar provides built-in security, resiliency and observability capabilities.

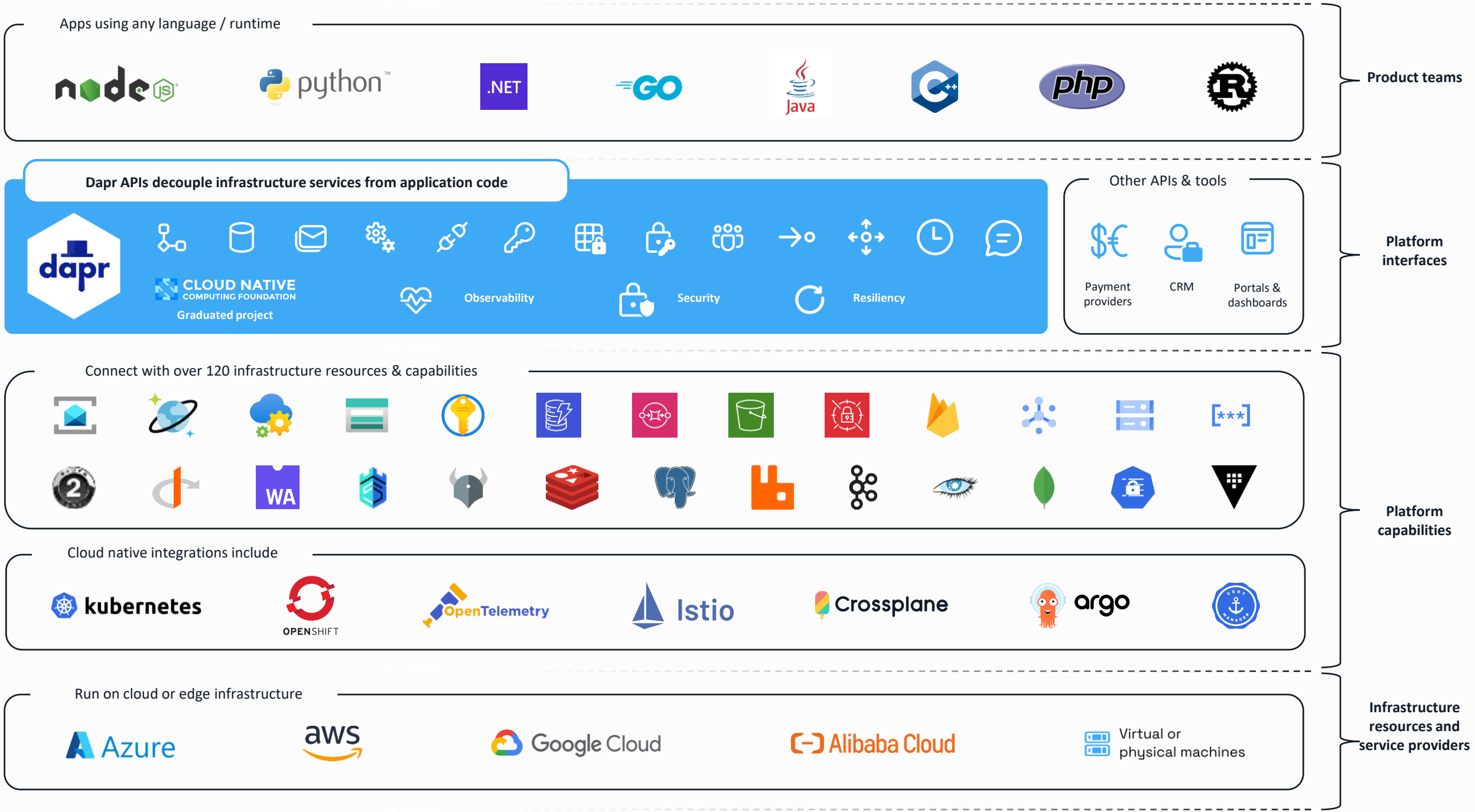
Speeds up application development by providing an integrated set of APIs for communication, state, and workflow.

# Dapr Goals

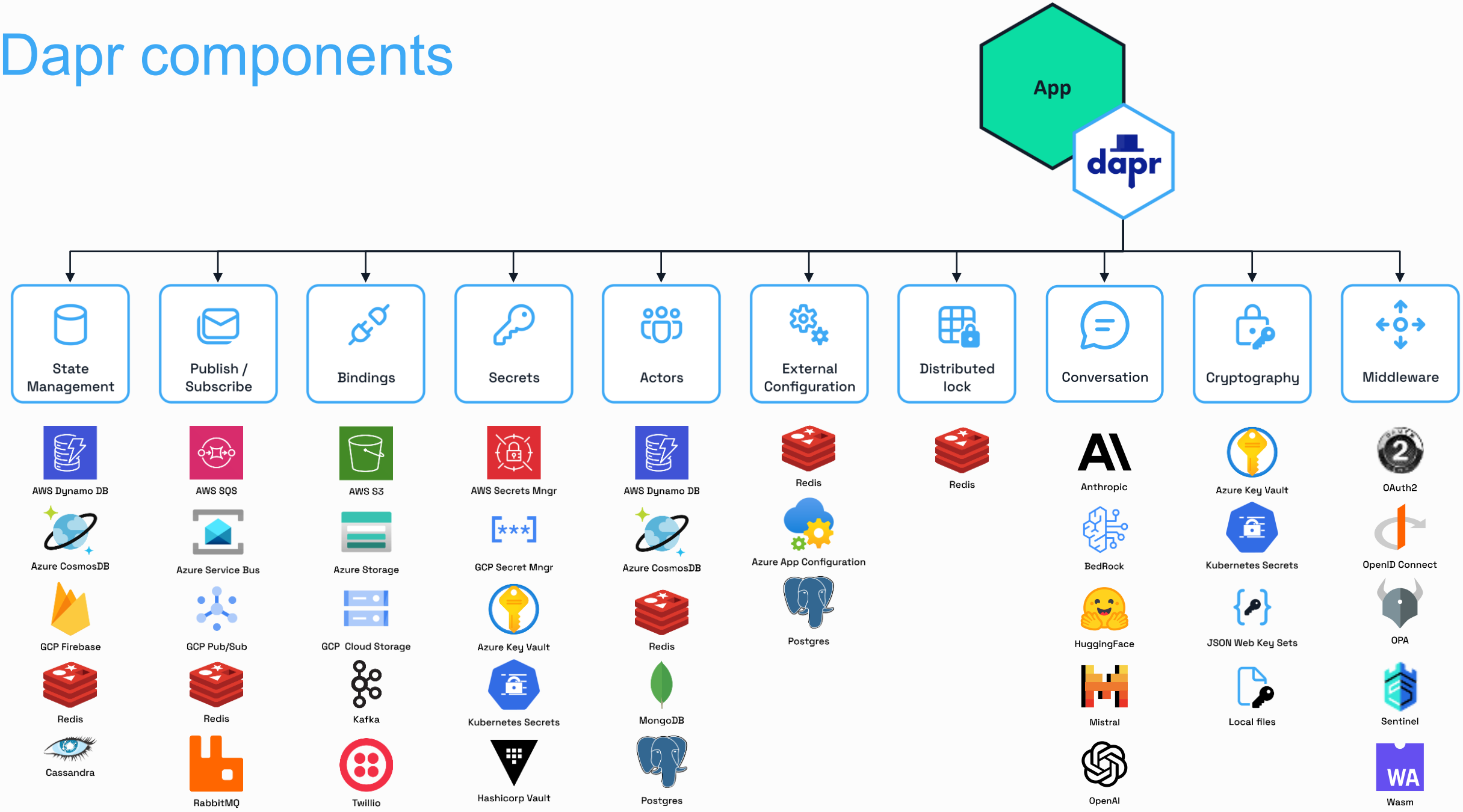
- Provide an integrated set of APIs
- Any language or framework
- Includes best practices & standards
- Platform agnostic
- Extensible and pluggable
- Community driven, vendor neutral

# Dapr – Application developer platform

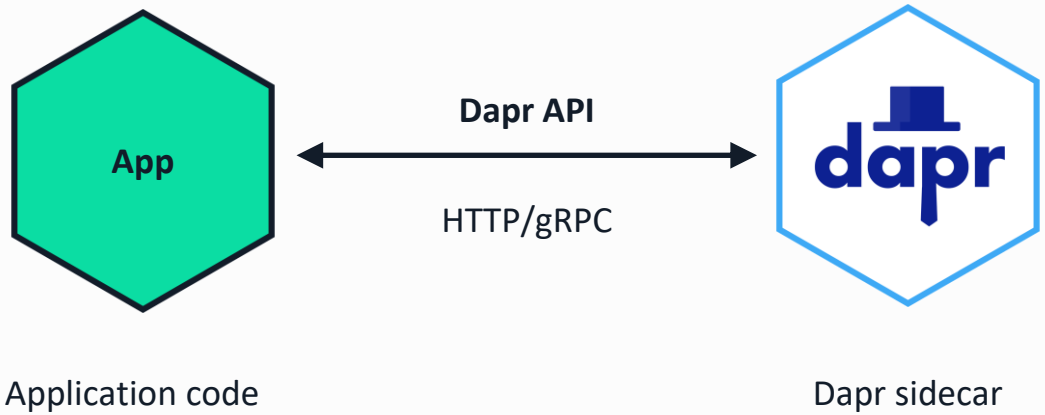



Apps using any language / runtime
node
python
.NET
GO
Java
C++
php
R
Product teams
Dapr APIs decouple infrastructure services from application code
dapr
CLOUD NATIVE COMPUTING FOUNDATION
Graduated project
Observability
Security
Resiliency
Other APIs & tools
Payment providers
CRM
Portals & dashboards
Platform interfaces
Connect with over 120 infrastructure resources & capabilities
WA
Platform capabilities
Cloud native integrations include
kubernetes
OPENSHIFT
OpenTelemetry
Istio
Crossplane
argo
Run on cloud or edge infrastructure
Azure
aws
Google Cloud
Alibaba Cloud
Virtual or physical machines
Infrastructure resources and service providers

# Dapr components

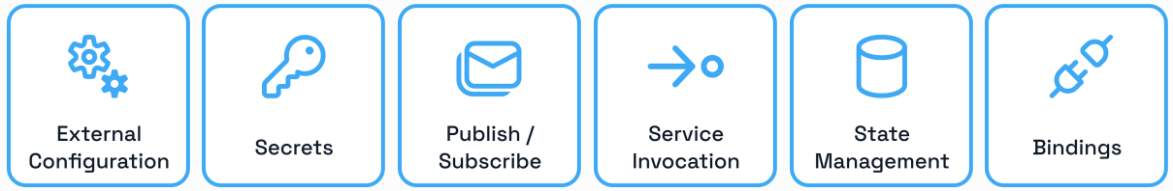


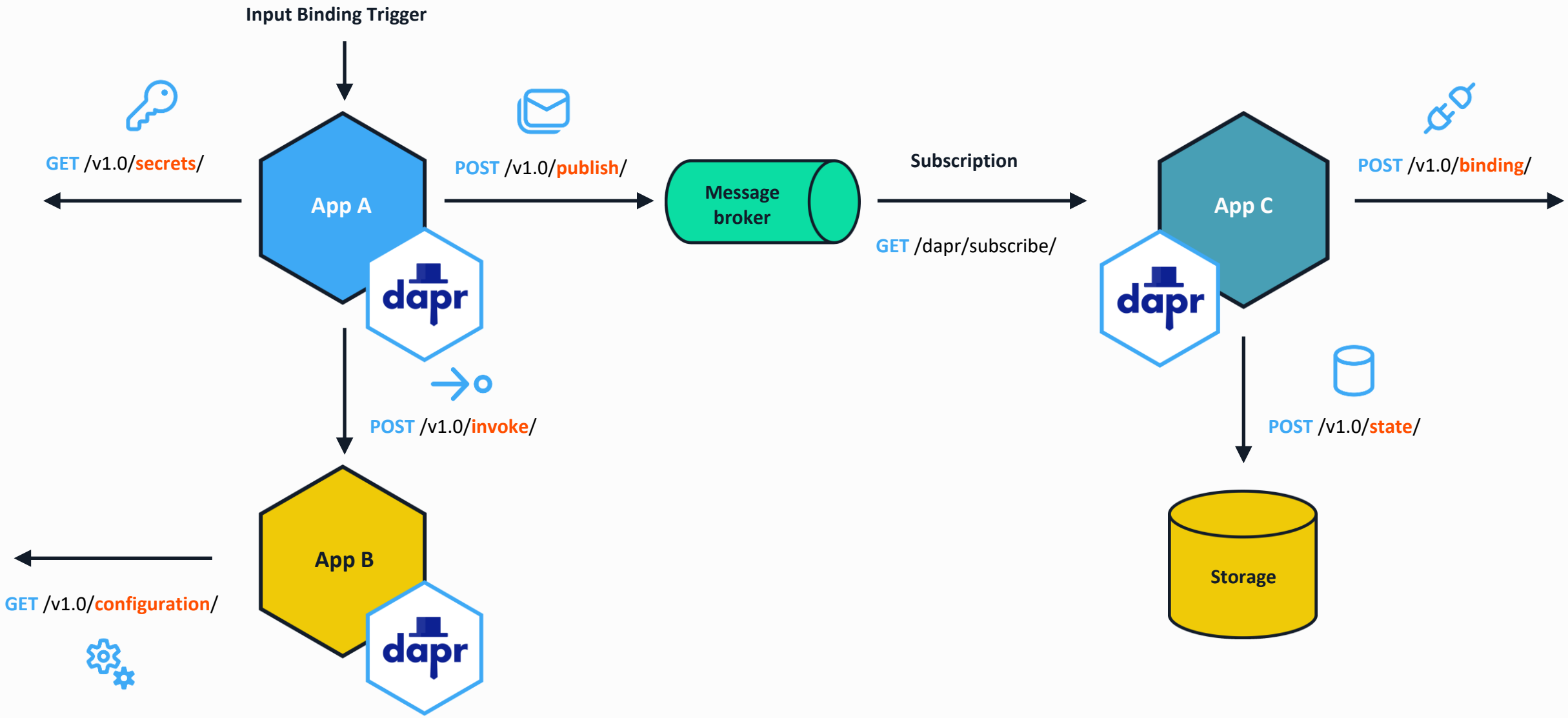
# Sidecar pattern and the Dapr API



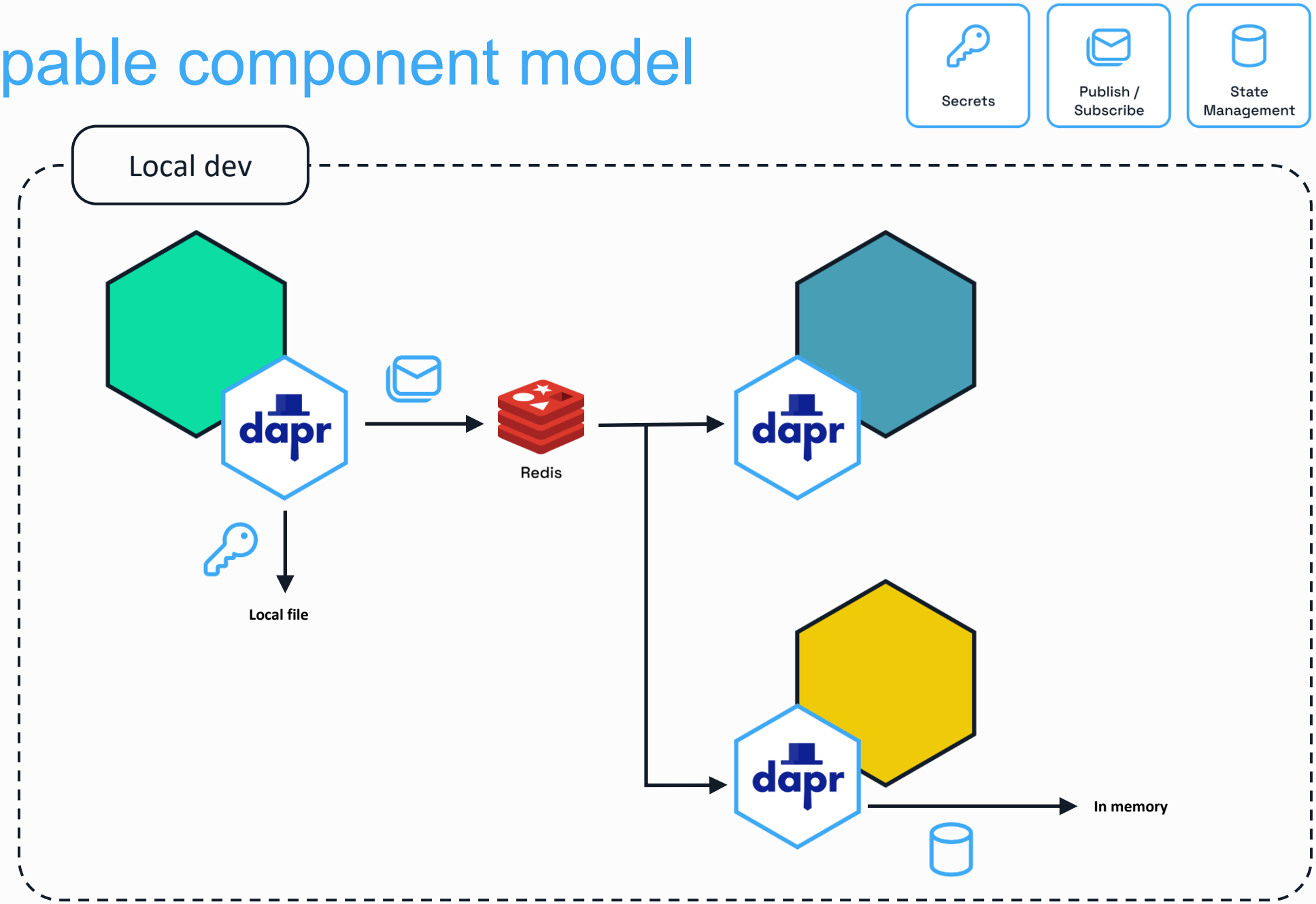

| | |
|---|---|
| **POST** | http://localhost:3500/v1.0/**invoke**/cart/method/order |
| **GET** | http://localhost:3500/v1.0/**state**/inventory/item50 |
| **POST** | http://localhost:3500/v1.0/**publish**/mybroker/order-messages |
| **GET** | http://localhost:3500/v1.0/**secrets**/vault/dbaccess |
| **POST** | http://localhost:3500/v1.0/**workflows**/dapr/businessprocess/start |

# Using the Dapr APIs

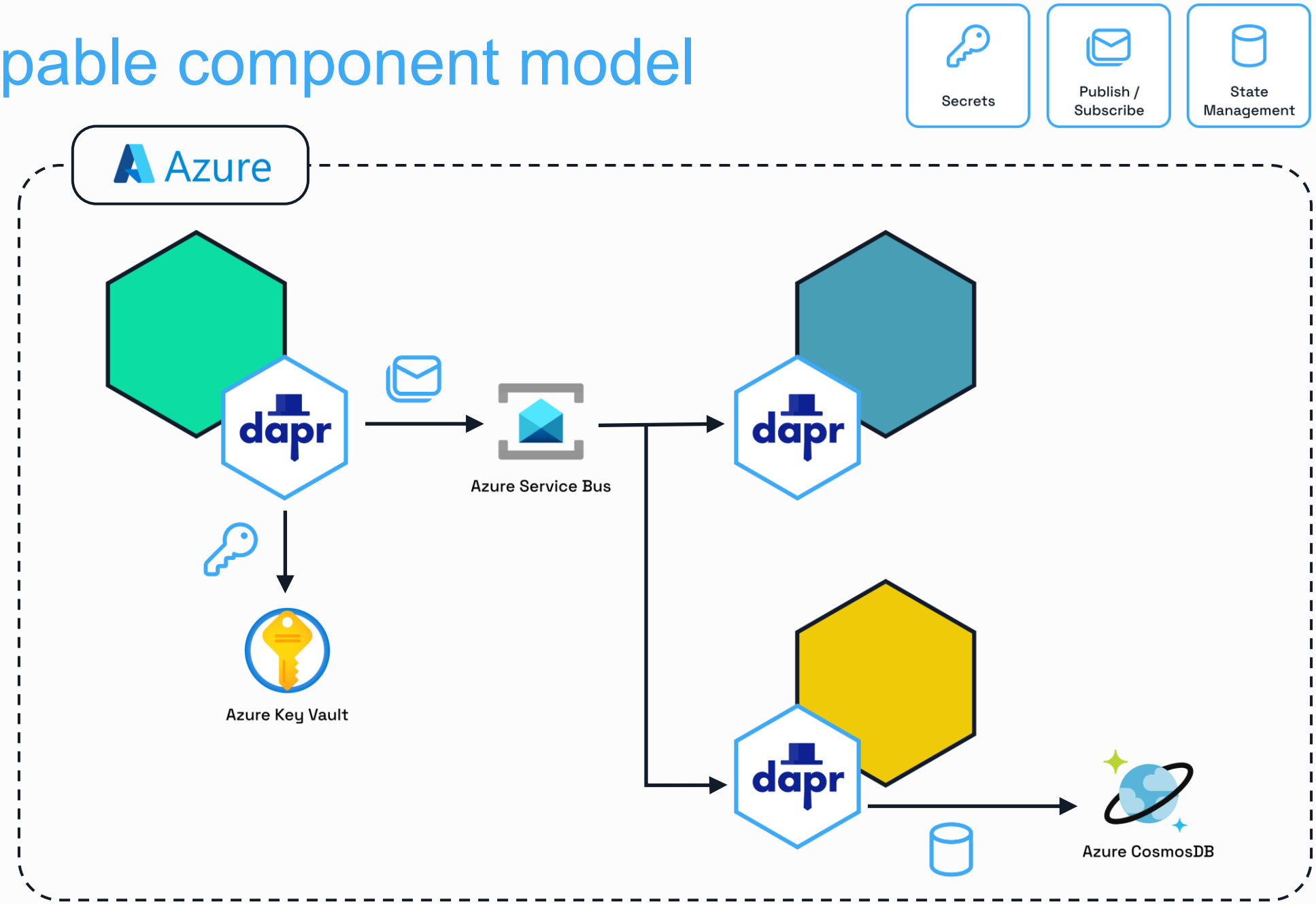


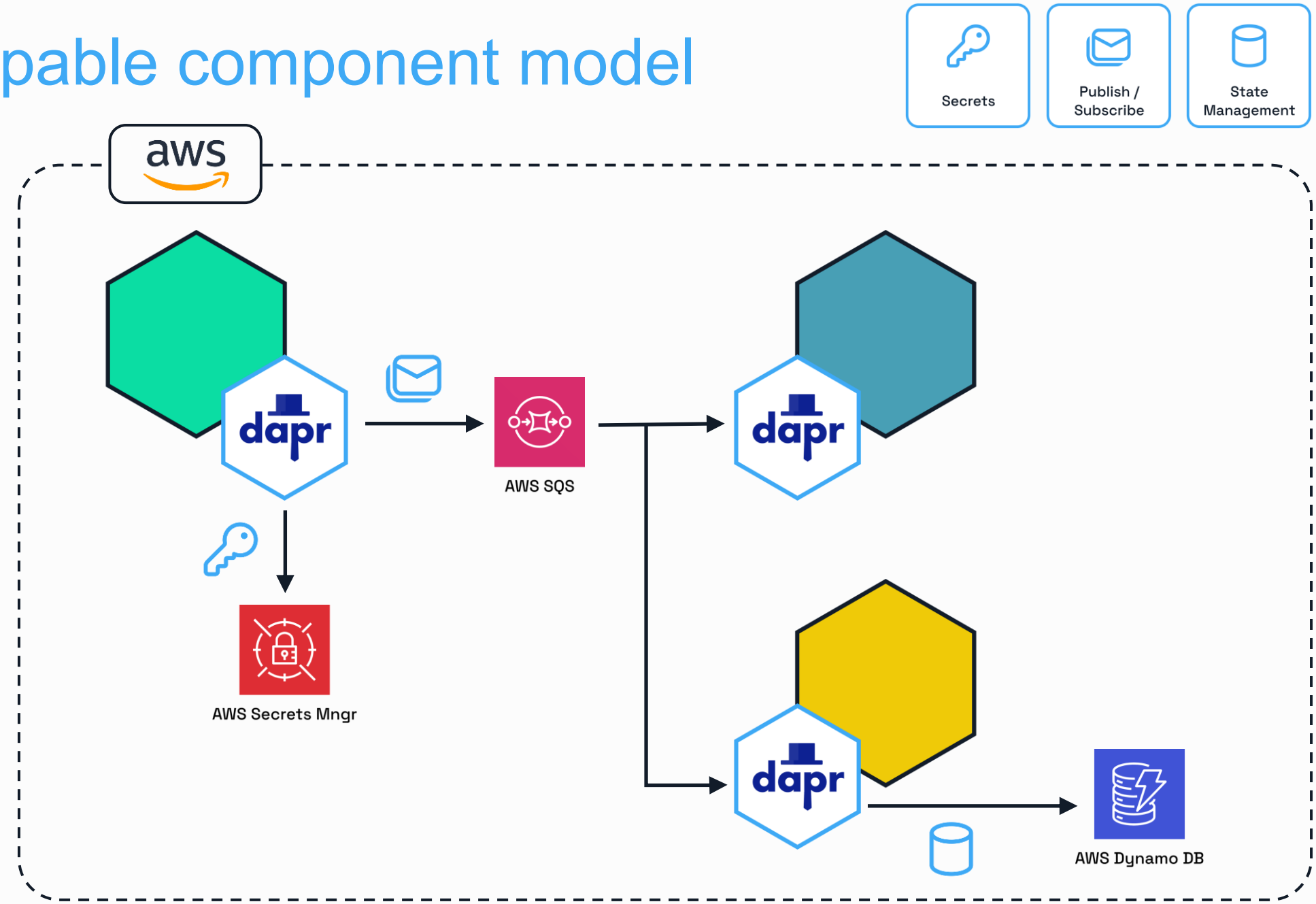
# Swappable component model



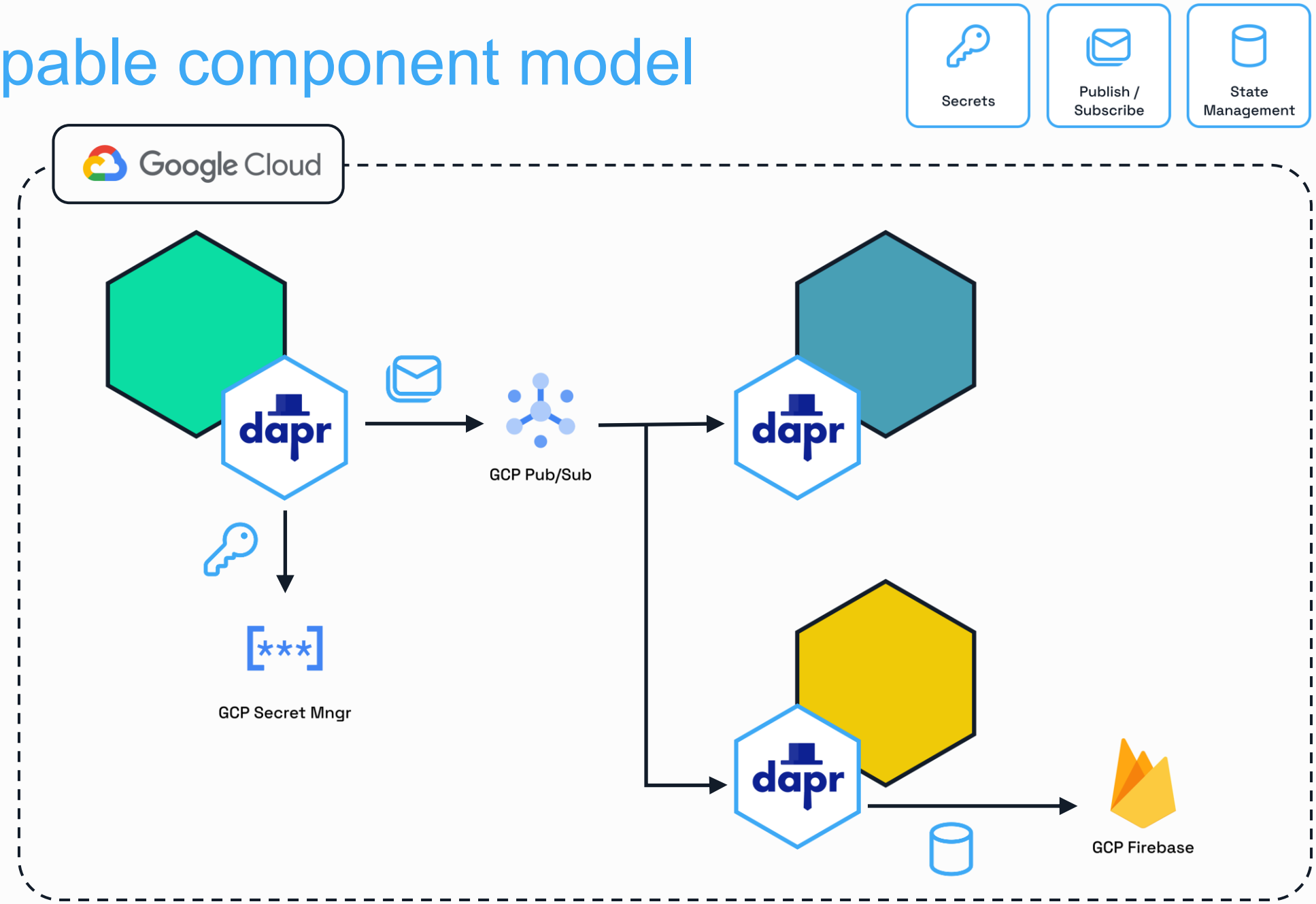
# Swappable component model



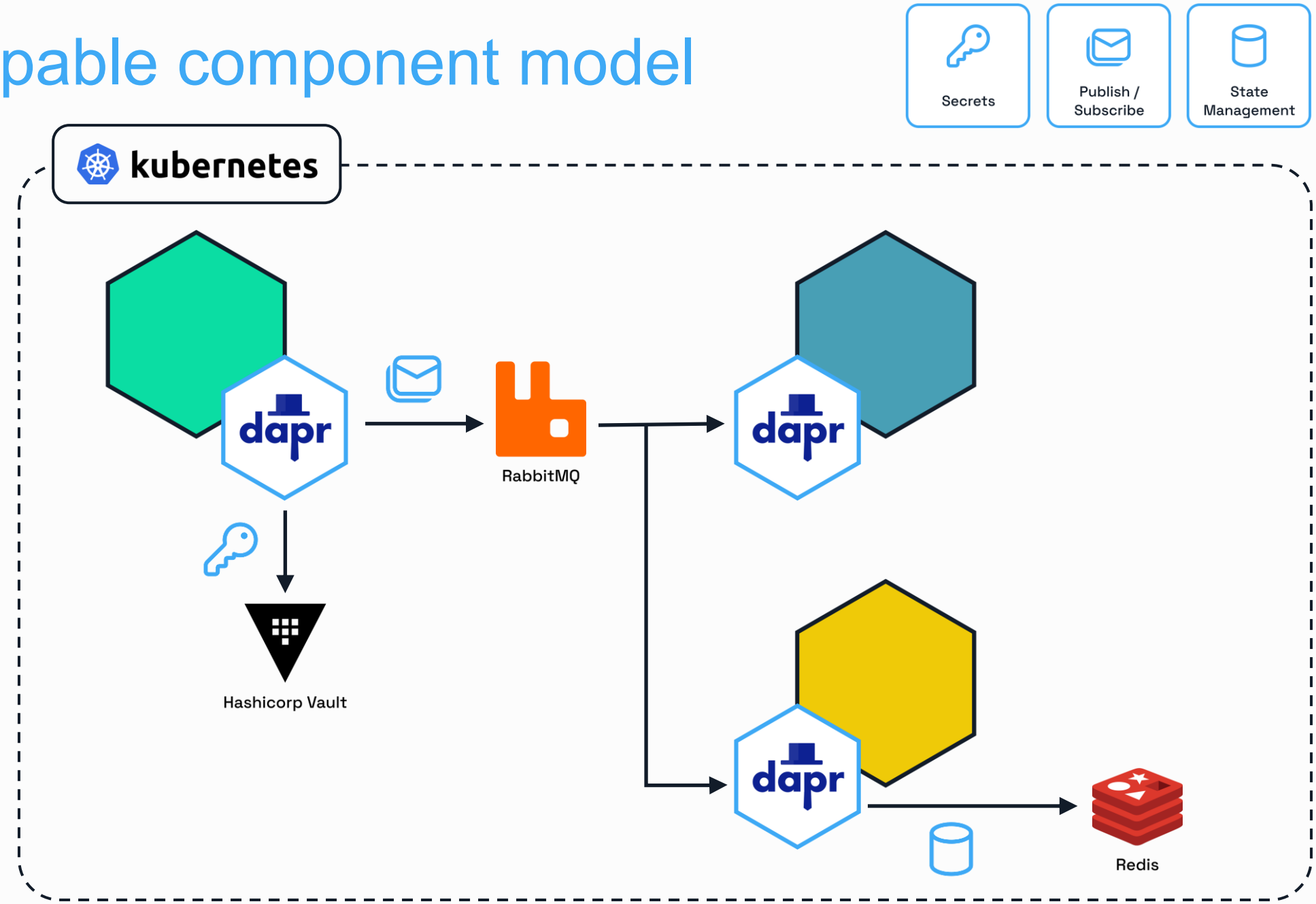
# Swappable component model



# Swappable component model



# Swappable component model



# Use Dapr anywhere

Azure Container Apps

Diagrid Catalyst

Microsoft Azure

Google Cloud

aws

Alibaba Cloud

kubernetes

Virtual or
physical machines

# Dapr community

**14/157**
Largest CNCF project

**120**
Community components

**700k+**
Docker Hub pulls/month

**300k**
Unique doc views/month

**31k**
GitHub stars

**3.9k**
Contributors

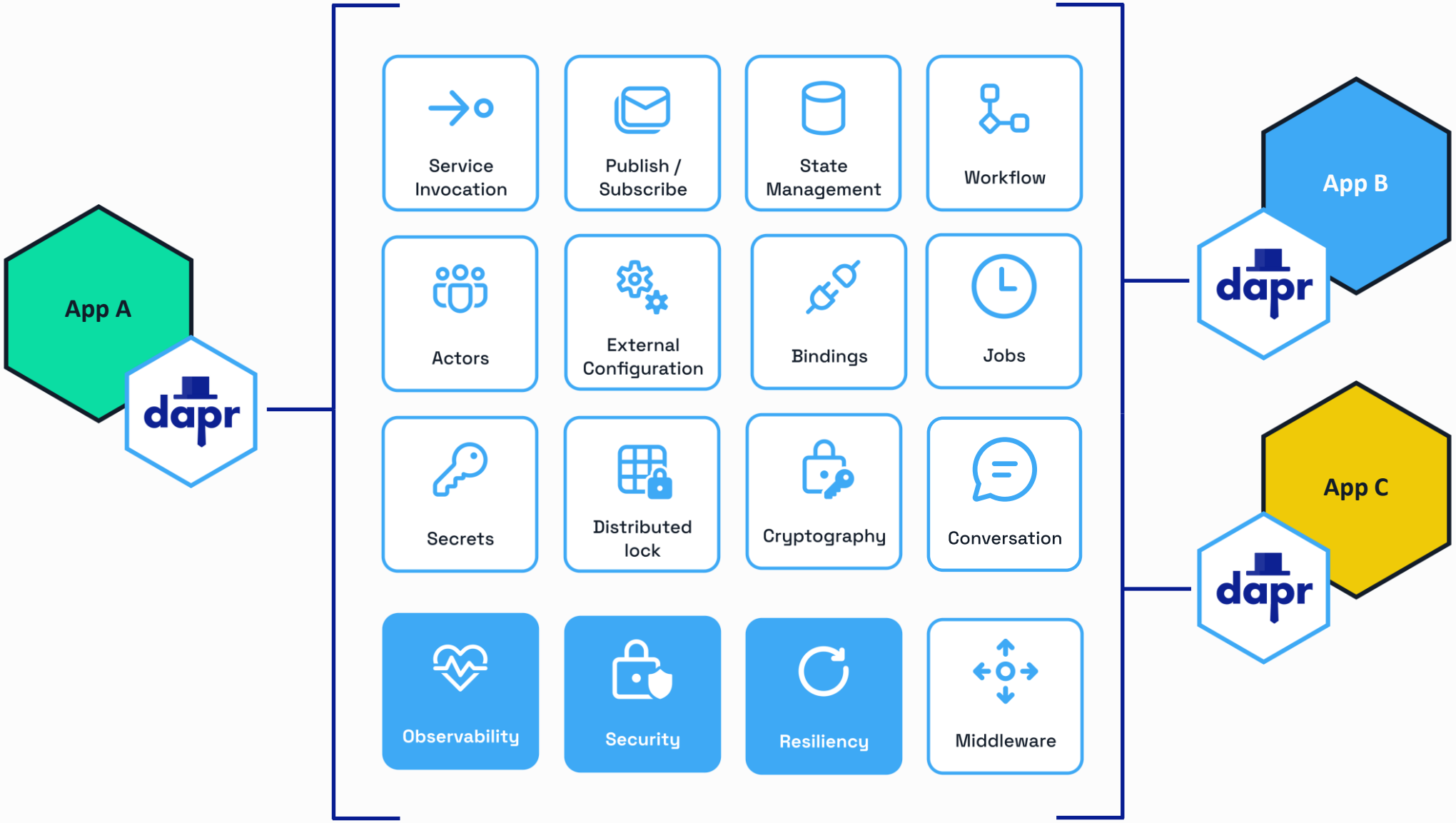
**8k**
Discord members

# Dapr contributors

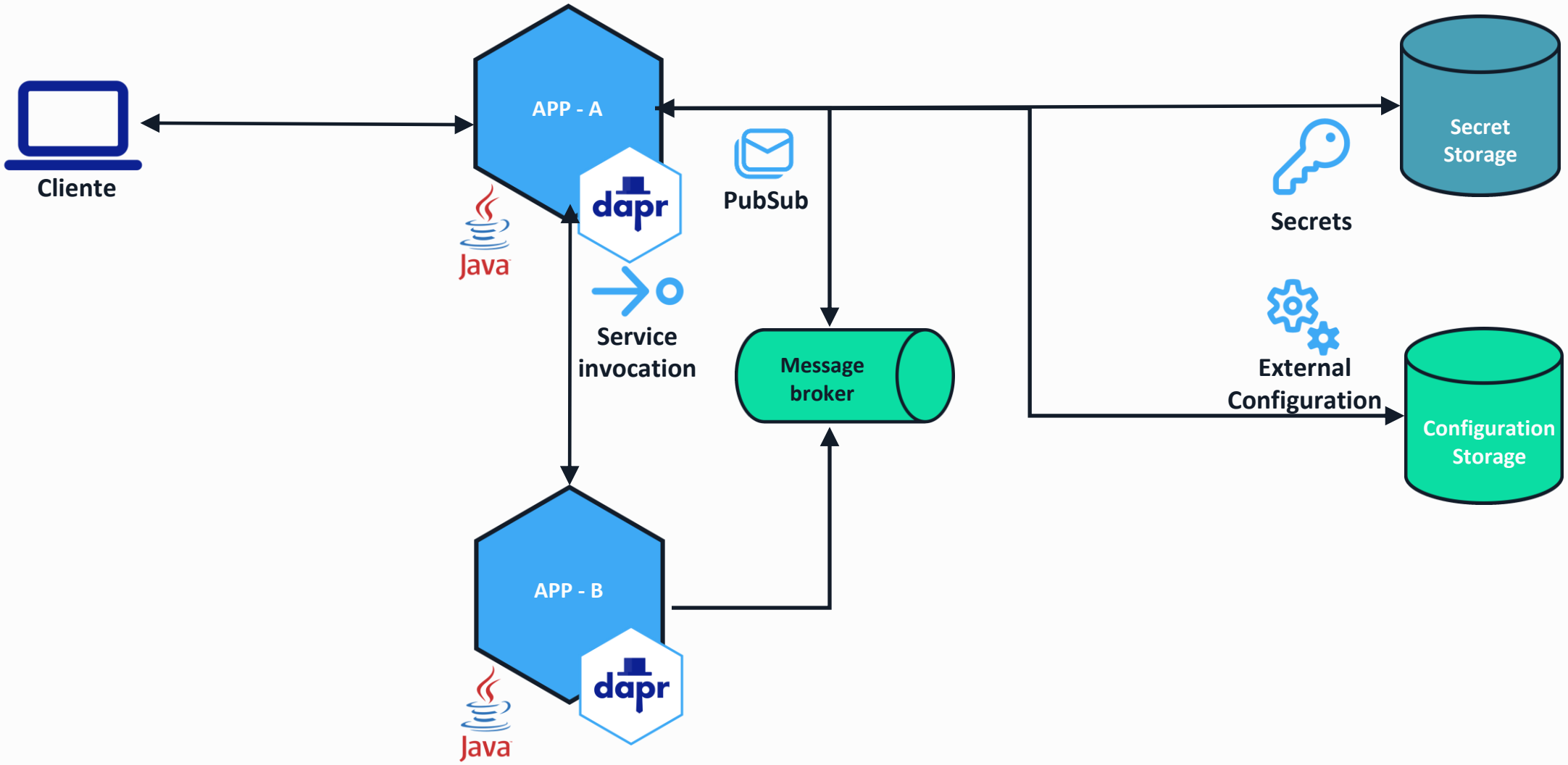
# Dapr users

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

# Dapr APIs & Cross cutting concerns



# DEMO


Cliente
APP - A
Java
dapr
Service
invocation
PubSub
Message
broker
Secrets
Secret
Storage
External
Configuration
Configuration
Storage
APP - B
Java
dapr

# DEMO

https://github.com/waltercoan/soujava2025-dapr


# 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

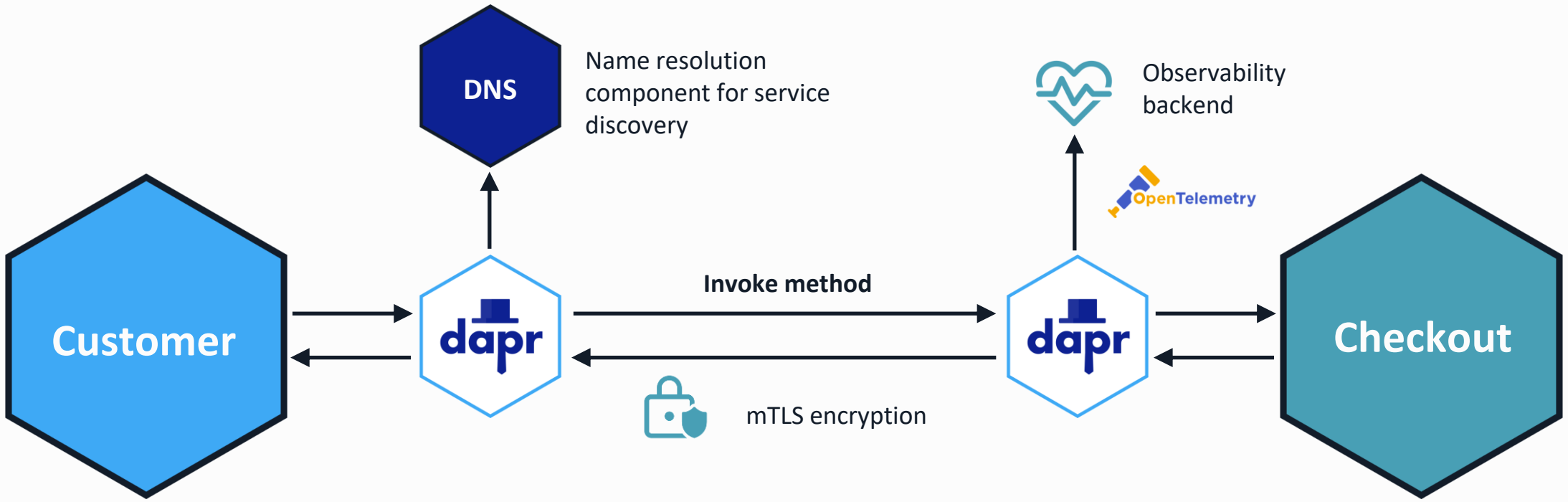
# Service Invocation




**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 in .NET


```
var order = new Order(orderId);

var client = DaprClient.CreateInvokeHttpClient(appId: "order-processor");

var response = await client.PostAsJsonAsync("/orders", order);
```

# Service Invocation in Python


```
base_url = os.getenv('BASE_URL', 'http://localhost') + ':' +
           os.getenv('DAPR_HTTP_PORT', '3500')

headers = {'dapr-app-id': 'order-processor', 'content-type': 'application/json'}

order = {'orderId': orderId}

result = requests.post(
        url='%s/orders' % (base_url),
        data=json.dumps(order),
        headers=headers
```


# Service Invocation Demo

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


# Publish / Subscribe API

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

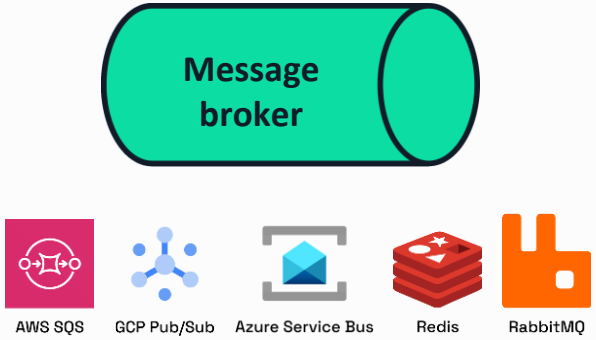
# Publish / Subscribe


**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



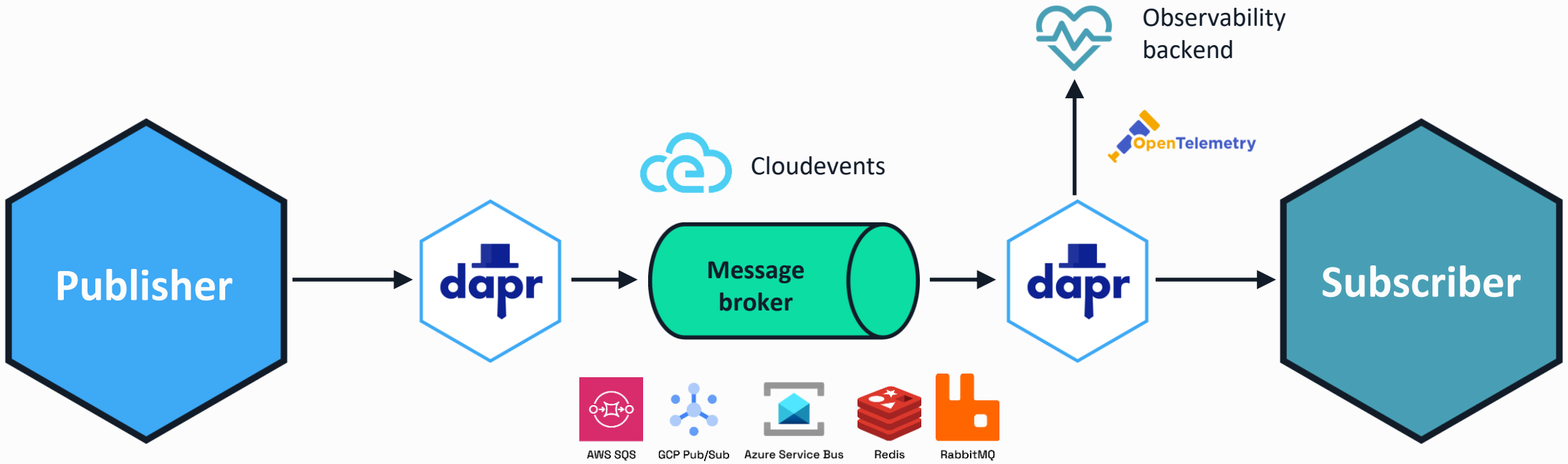




**POST**
http://localhost:3500/v1.0/**publish**/mybroker/order-messages

**POST**
http://localhost:5100/orders

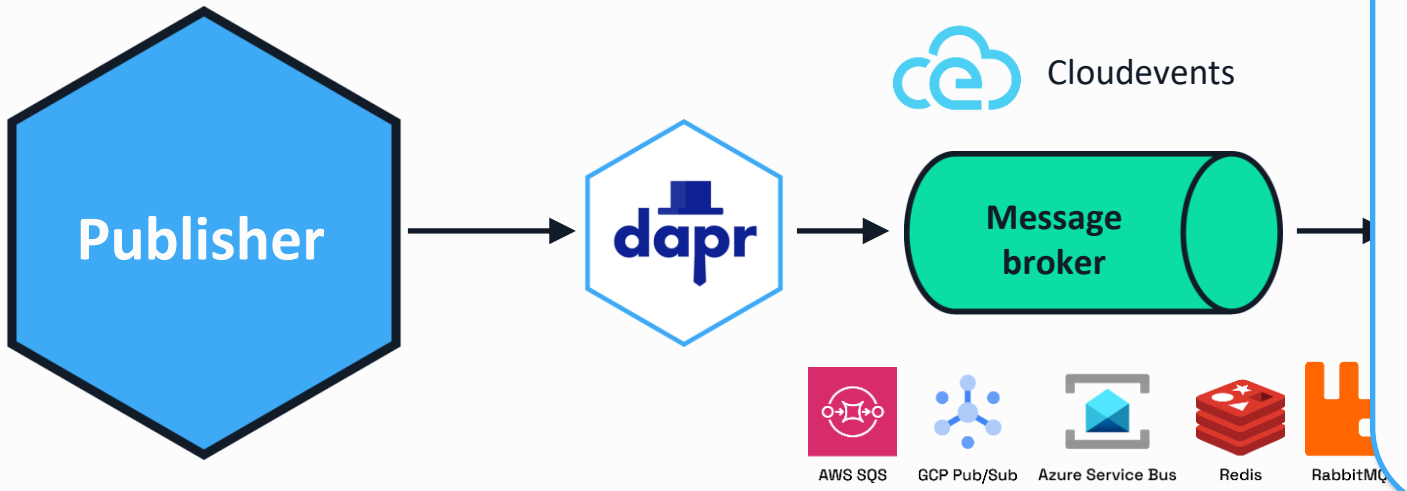
# Publish / Subscribe


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

**POST**
http://localhost:5100/orders

# Publish / Subscribe Component


```yaml
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 with .NET SDK


**Publish**

```
var order = new Order(orderId);

using var client = new DaprClientBuilder().Build();

await client.PublishEventAsync("orderpubsub", "orders", order);
```

**Subscribe**

```
app.UseCloudEvents();

app.MapSubscribeHandler();

app.MapPost("/orders", [Topic("orderpubsub", "orders")] (Order order) =>
{
    return Results.Ok(order);
});
```

# Publish / Subscribe with Python SDK


**Publish**

```
with DaprClient() as client:
    order = {'orderId': orderId}

    result = client.publish_event(
            pubsub_name='orderpubsub',
            topic_name='orders',
            data=json.dumps(order),
            data_content_type='application/json',
    )
```

# Publish / Subscribe with Python SDK


**Subscribe**

```
@app.route('/dapr/subscribe', methods=['GET'])
def subscribe():
    subscriptions = [{
        'pubsubname': 'orderpubsub',
        'topic': 'orders',
        'route': 'orders'
    }]
    return jsonify(subscriptions)

@app.route('/orders', methods=['POST'])
def orders_subscriber():
    event = from_http(request.headers, request.get_data())
    return json.dumps({'success': True}), 200, {
        'ContentType': 'application/json'}
```


# Publish / Subscribe Demo

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


# State Management API (Key/Value)

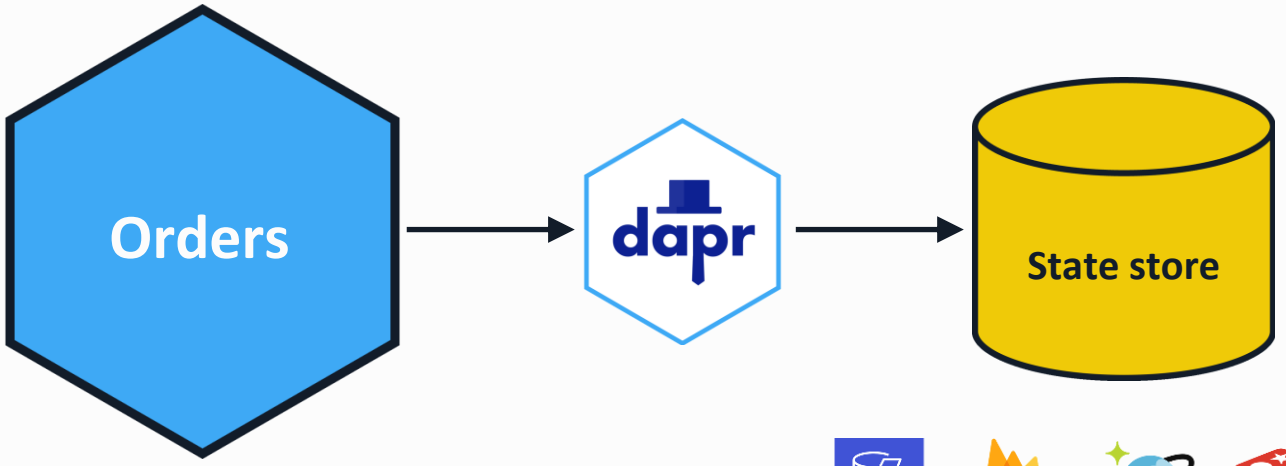
https://docs.dapr.io/developing-applications/building-blocks/state-management/

# 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)

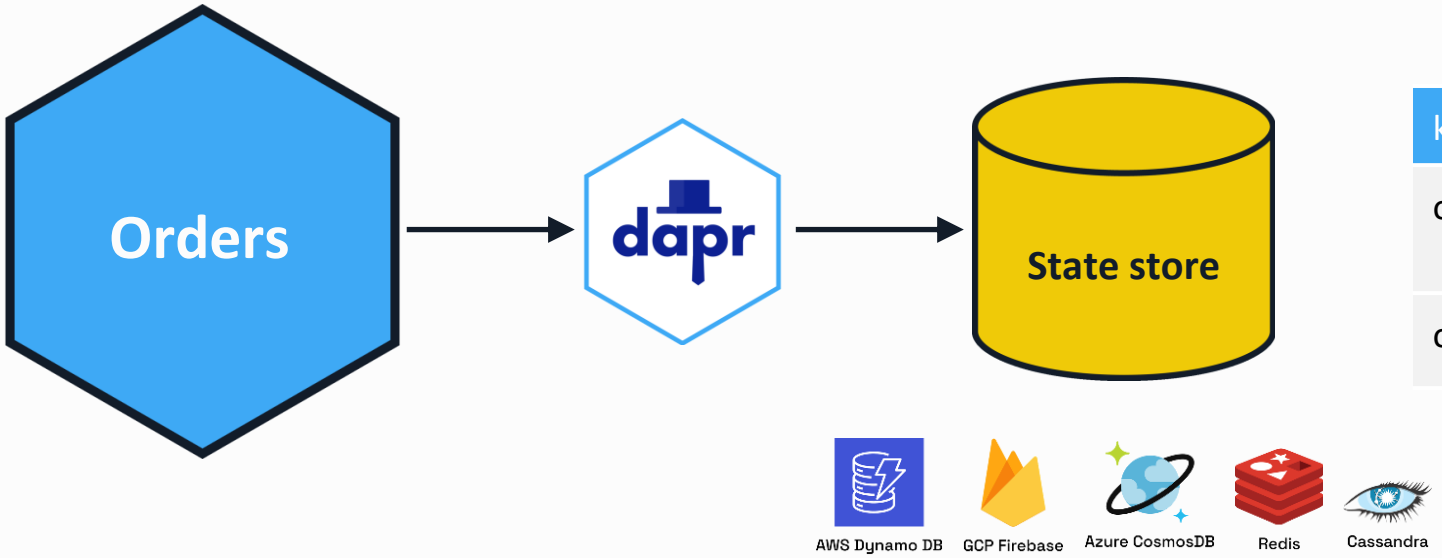


| key | field | value |
|---|---|---|
| orders\|\|order1 | data | "{orderId:1}" |
| orders\|\|order1 | version | 1 |

**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 with .NET SDK


```
var order = new Order(orderId);

await client.SaveStateAsync(
        DAPR_STORE_NAME,
        orderId.ToString(),
        order.ToString());

var result = await client.GetStateAsync<string>(
        DAPR_STORE_NAME,
        orderId.ToString());

await client.DeleteStateAsync(
        DAPR_STORE_NAME,
        orderId.ToString());
```

# State Management with Python SDK


```python
with DaprClient() as client:
        order = {'orderId': orderId}

        client.save_state(DAPR_STORE_NAME, orderId, str(order))

        result = client.get_state(DAPR_STORE_NAME, orderId)

        client.delete_state(store_name=DAPR_STORE_NAME, key=orderId)
```


# State Management (Key/Value) Demo

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


# External Configuration API

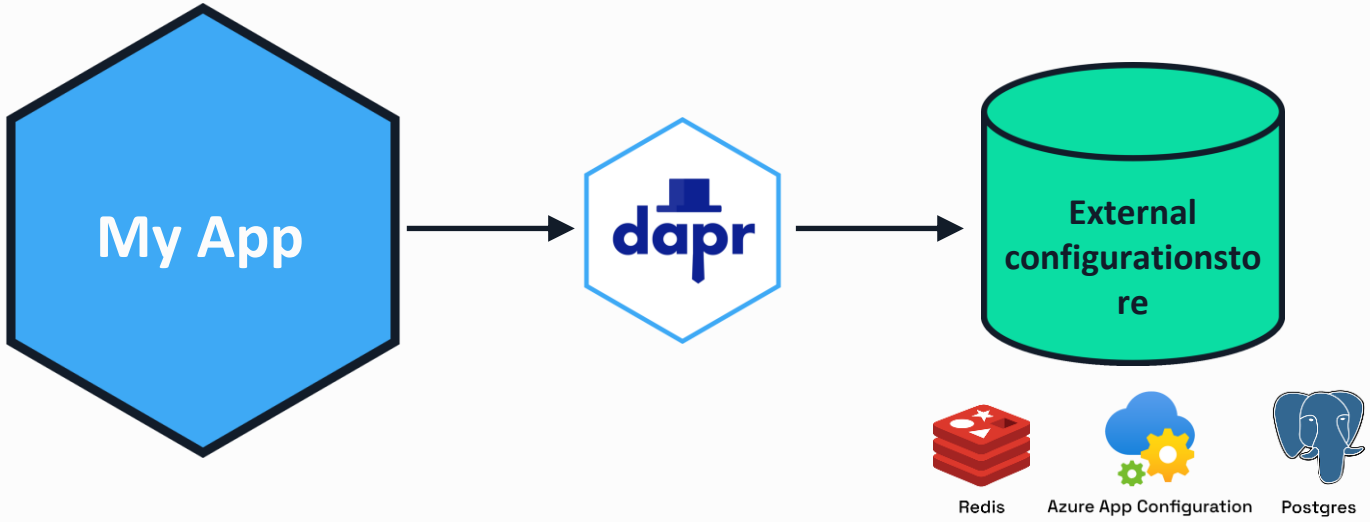
https://docs.dapr.io/developing-applications/building-blocks/configuration/

# External configuration


**The external configuration API enables read access to configuration data.**

- Integrates with many configuration stores
- Subscribe to configuration changes
- Use scopes to limit access

# External Configuration


GET http://localhost:3500/v1.0/configuration/myconfig?key=config1

RESPONSE

```
{
  “config1”: {
    “value” : “configvalue”
  }
}
```

# External Configuration with .NET SDK


```
const string DAPR_CONFIGURATION_STORE = "configstore";
var CONFIGURATION_ITEMS = new List<string> { "orderId1", "orderId2" };

var client = new DaprClientBuilder().Build();

var config = await client.GetConfiguration(
        DAPR_CONFIGURATION_STORE,
        CONFIGURATION_ITEMS);

foreach (var item in config.Items)
{
        var configItem = System.Text.Json.JsonSerializer.Serialize(item.Value);
}
```

# External Configuration with Python SDK


```python
DAPR_CONFIGURATION_STORE = 'configstore'
CONFIGURATION_KEYS = ['orderId1', 'orderId2']

with DaprClient() as client:
    for key in CONFIGURATION_KEYS:
        resp = client.get_configuration(store_name=DAPR_CONFIGURATION_STORE,
               keys=[key], config_metadata={})
        print(f"Configuration for {key} : {resp.items[key].value}", flush=True)
```


# External Configuration Demo

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


# Secrets Management API

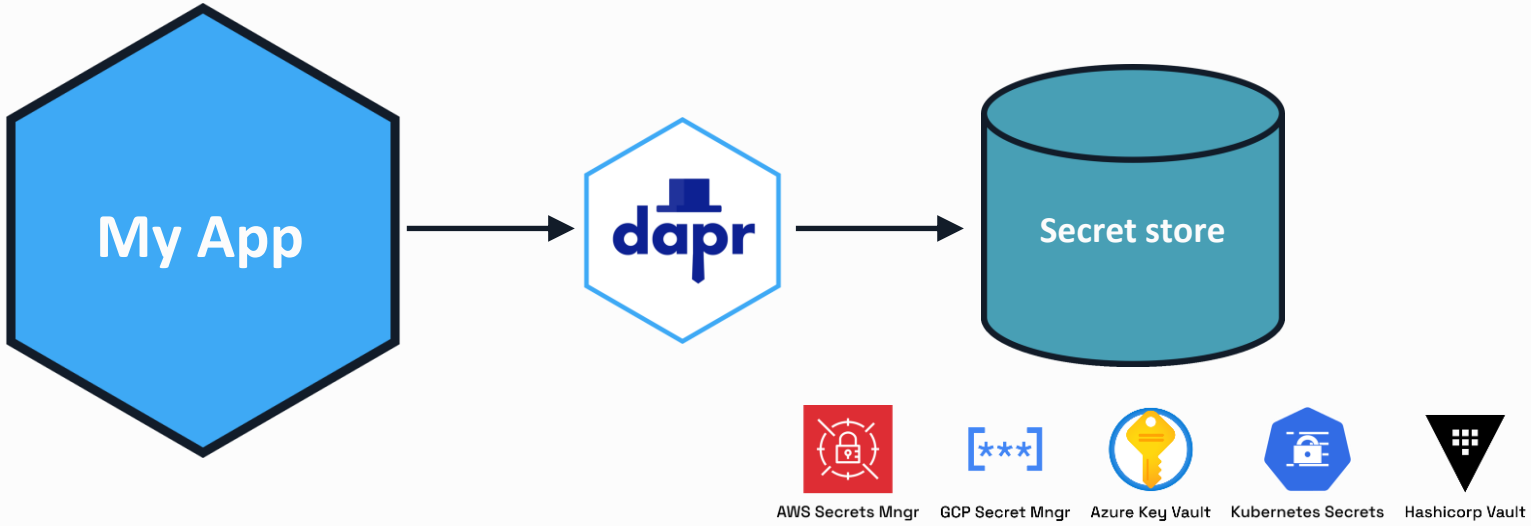
https://docs.dapr.io/developing-applications/building-blocks/secrets/secrets-overview/

# Secrets Management


**The secrets management API enables access to sensitive information in secret stores.**

- Integrates with public cloud and cloud-native secret stores
- Safely access secrets in your applications
- Reference secrets in Dapr components
- Use scopes to limit access

# Secrets Management


**GET** http://localhost:3500/v1.0/**secrets**/myvault/mysecret

RESPONSE

```
{
  "mysecret": "secretvalue"
}
```

# Secrets Management with .NET SDK


```
const string DAPR_SECRET_STORE = "localsecretstore";
const string SECRET_NAME = "secret";

var client = new DaprClientBuilder().Build();

var secret = await client.GetSecretAsync(DAPR_SECRET_STORE, SECRET_NAME);
```

# Secrets Management with Python SDK


```
DAPR_SECRET_STORE = 'localsecretstore'
SECRET_NAME = 'secret’

with DaprClient() as client:
        secret = client.get_secret(store_name=DAPR_SECRET_STORE, key=SECRET_NAME)
```


# Secrets Management Demo

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


# Security

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

# Secure Dapr to Dapr communication


# mTLS in K8s mode


Sentry Pod

Read existing root/issuer cert
OR
Create self-signed cert

Pod A

Pod B

(optional)
Operator brings existing
root and issues certs

Service A

Request workload certificate

Service B

Kubernetes Secrets

mTLS

Operator

Inject trust anchors (root
cert)

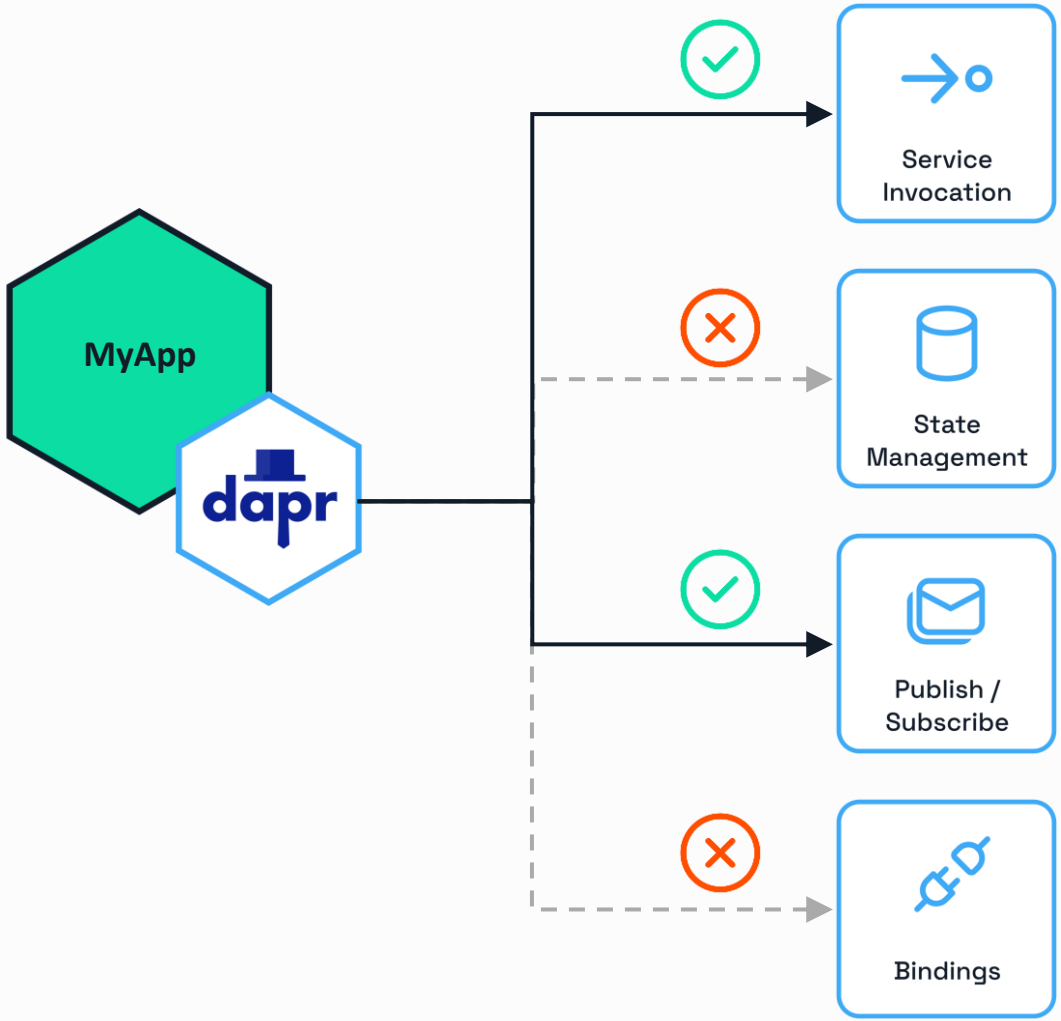
Injector Pod

Read root cert

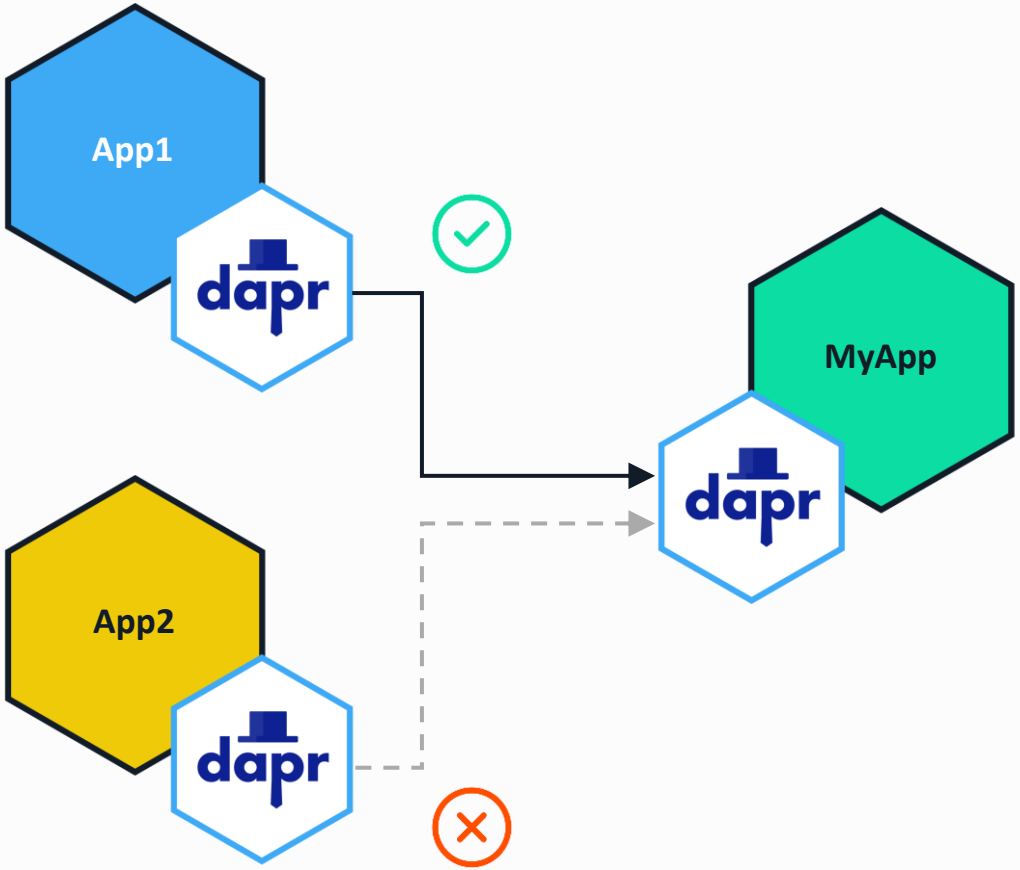
kubernetes

# Access policies


```yaml
apiVersion: dapr.io/v1alpha1
kind: Configuration
metadata:
  name: appconfig
  namespace: default
spec:
  api:
    allowed:
    - name: invoke
      version: v1
      protocol: grpc
    - name: pubsub
      version: v1
      protocol: grpc
```



# Access policies

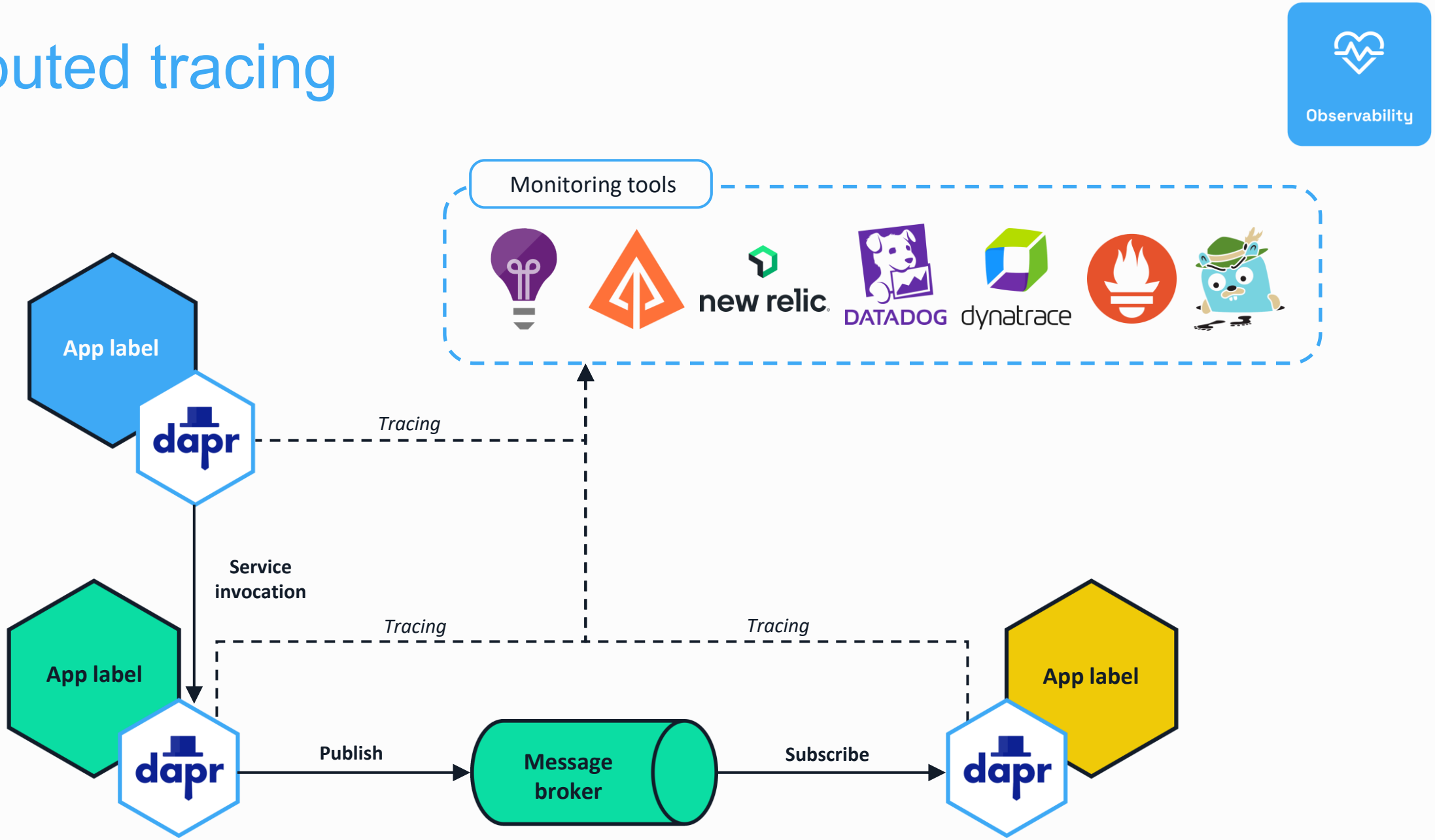




```yaml
apiVersion: dapr.io/v1alpha1
kind: Configuration
metadata:
  name: appconfig
spec:
  accessControl:
    defaultAction: deny
    trustDomain: "public"
    policies:
    - appId: app1
      defaultAction: allow
      trustDomain: 'public'
      namespace: "default"
```


# Observability

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

# Distributed tracing

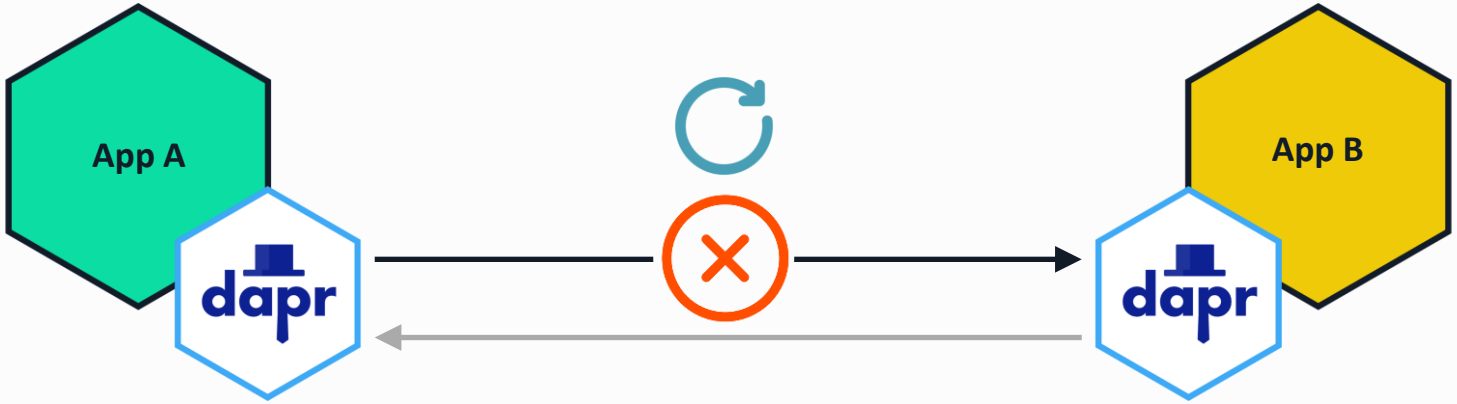



# Resiliency

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

# Service invocation resiliency

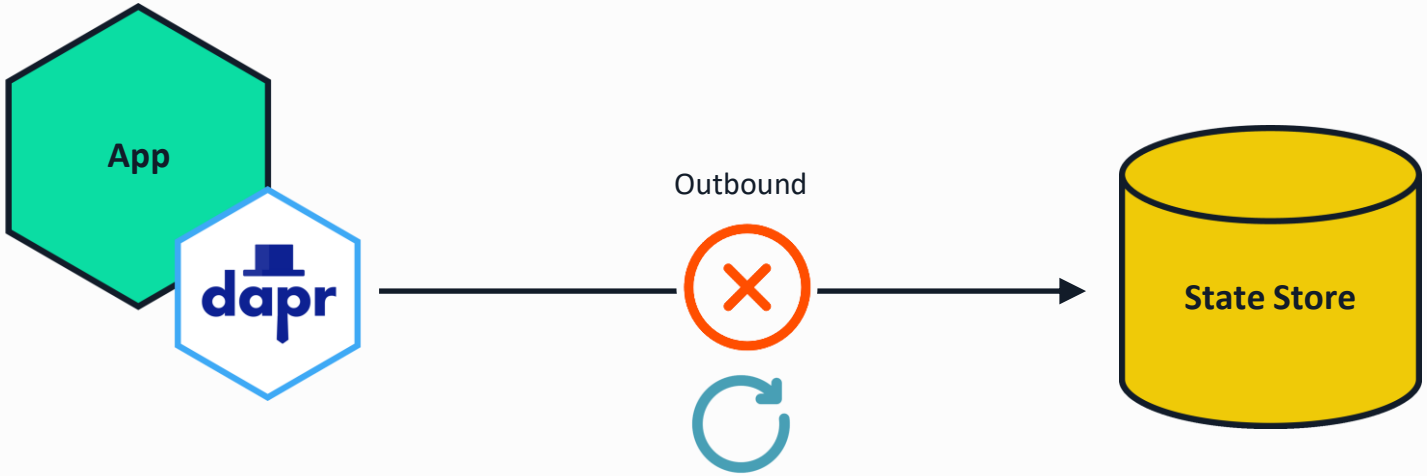

The built-in service invocation retries are always performed with a backoff interval of 1 second up to a threshold of 3 times.




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

# Outbound component resiliency

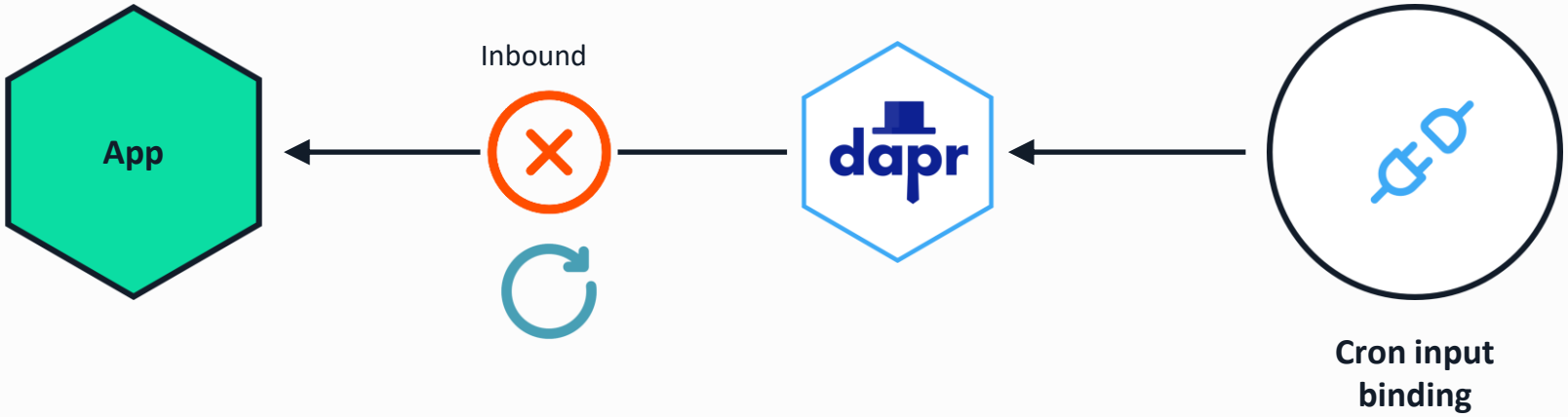

Component resiliency polices can be applied to outbound component calls. For example, calls to a state store.




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

# Inbound component resiliency

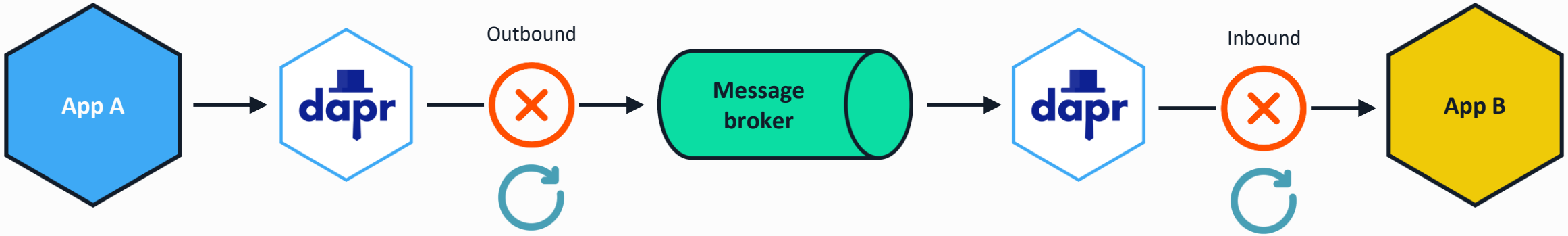

Resiliency polices can be applied to inbound component calls. For example, pub/sub subscriptions and input bindings.



# 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.

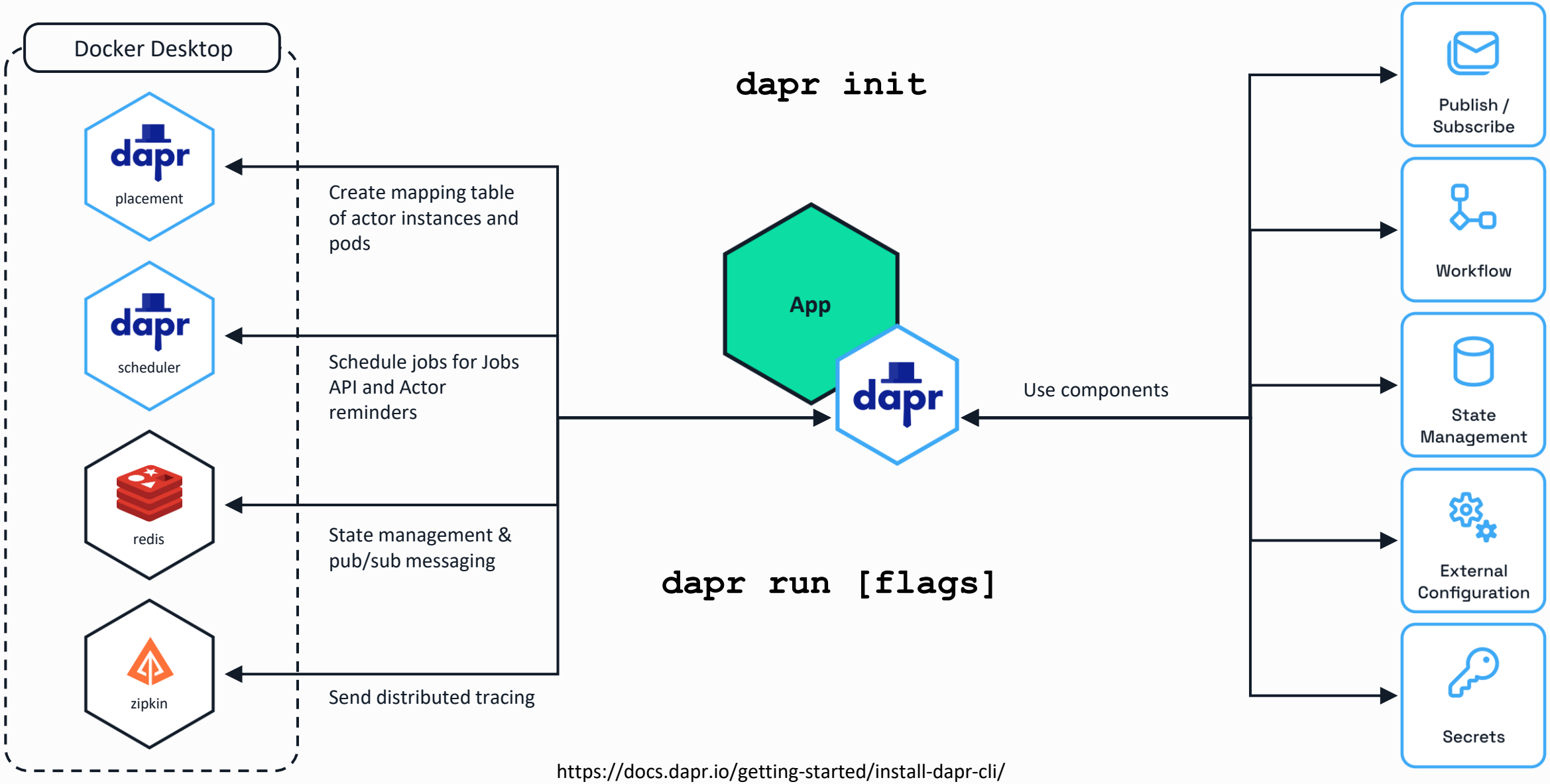
# Hosting modes

**Serverless**

The Dapr side car is hosted by a provider.

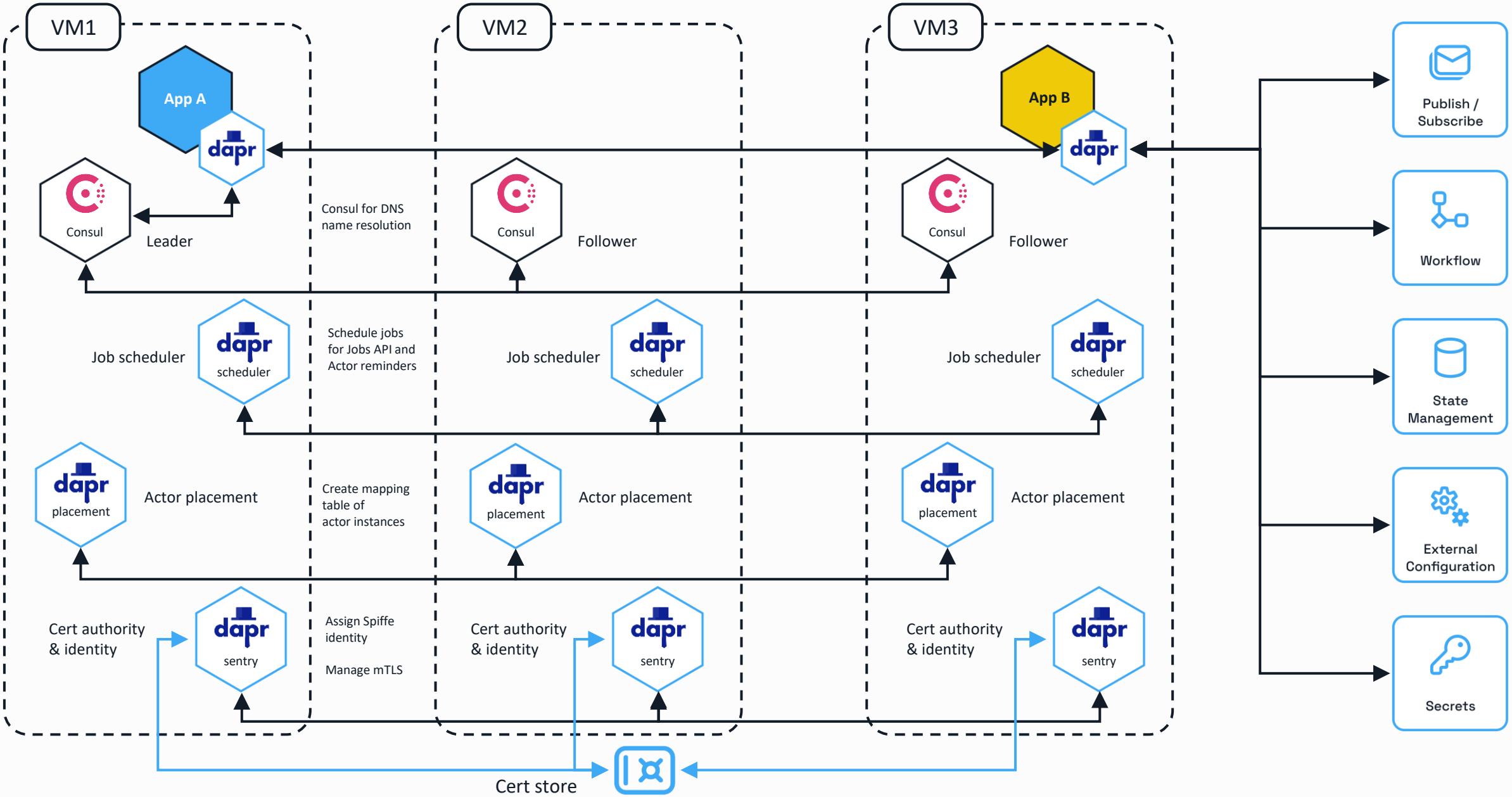
You only manage your applications.

# Local development with the Dapr CLI

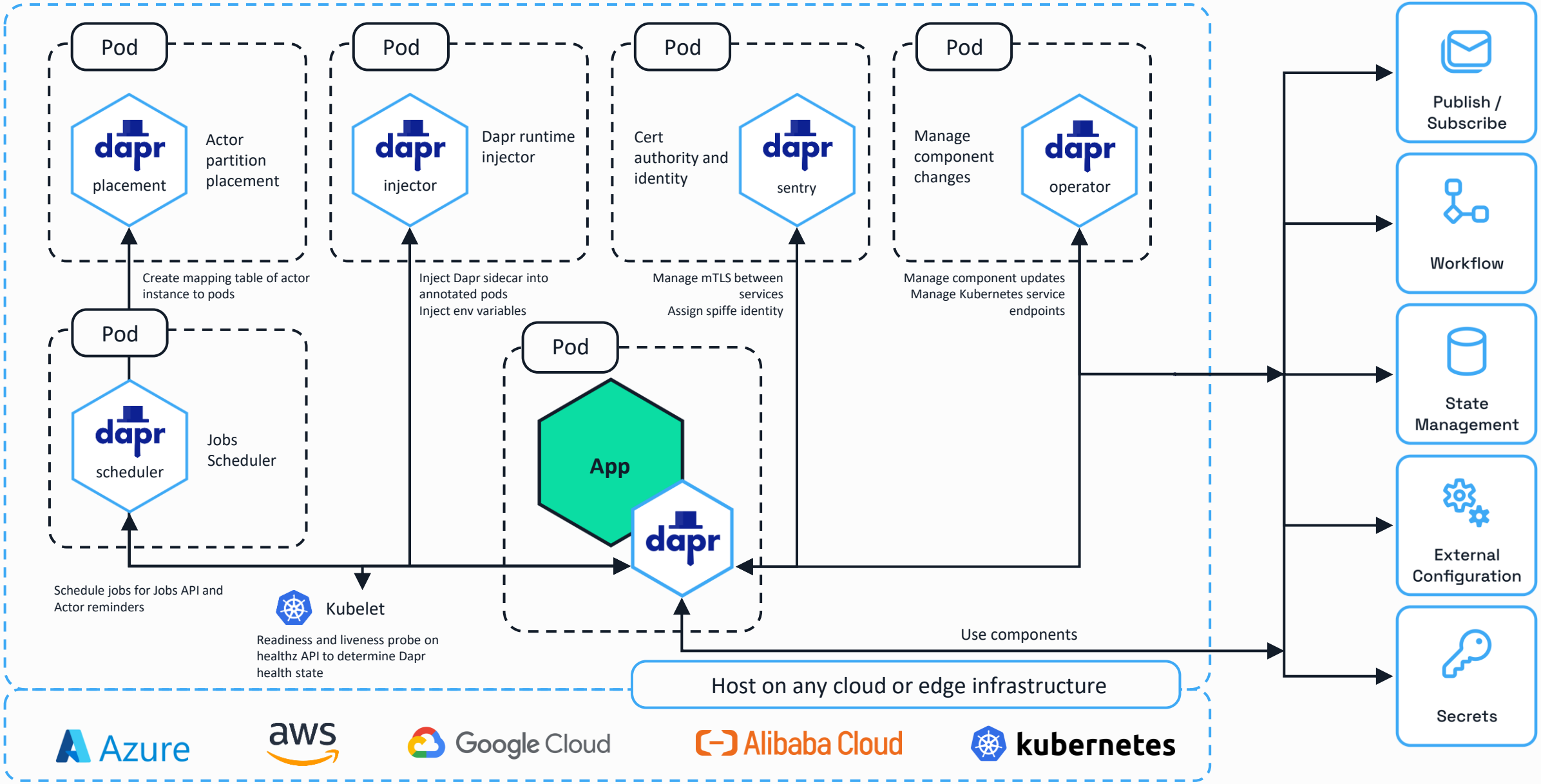



https://docs.dapr.io/getting-started/install-dapr-cli/

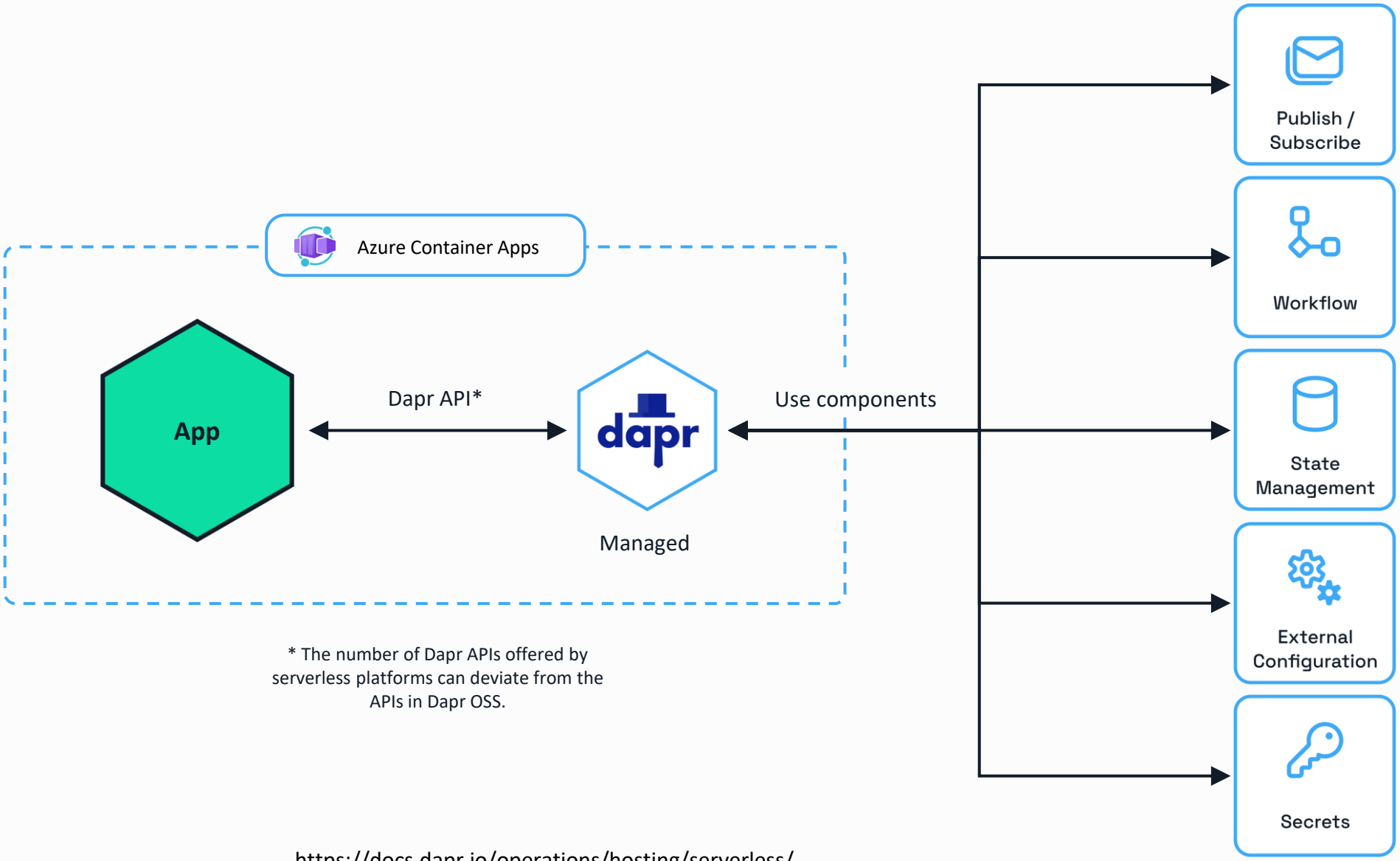
# Dapr in self-hosted mode on VMs



# Dapr on Kubernetes



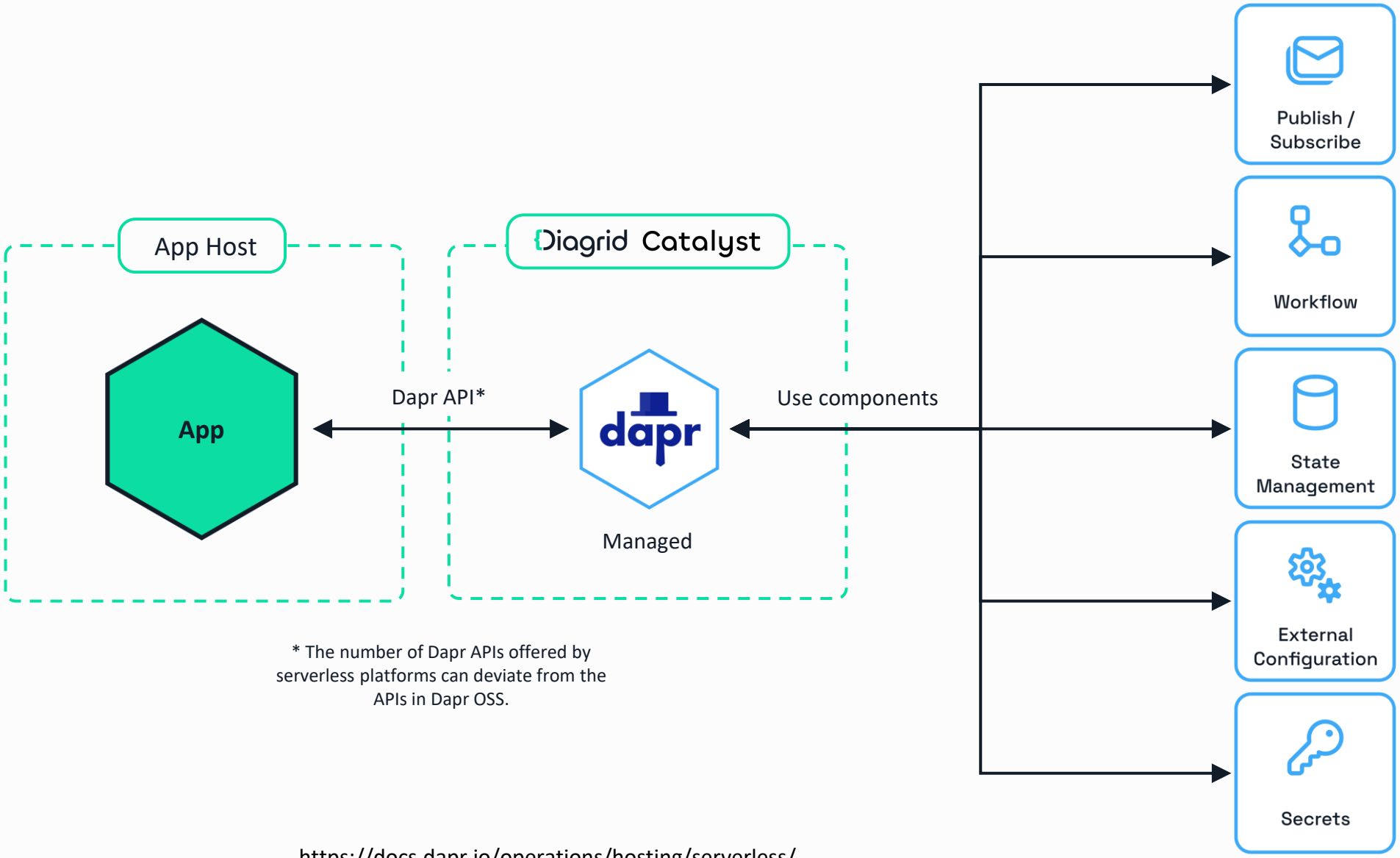
# Serverless




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

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

# 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

@daprdev.bsky.social

dapr.io

# Claim the Dapr Community Supporter badge!


bit.ly/dapr-supporter