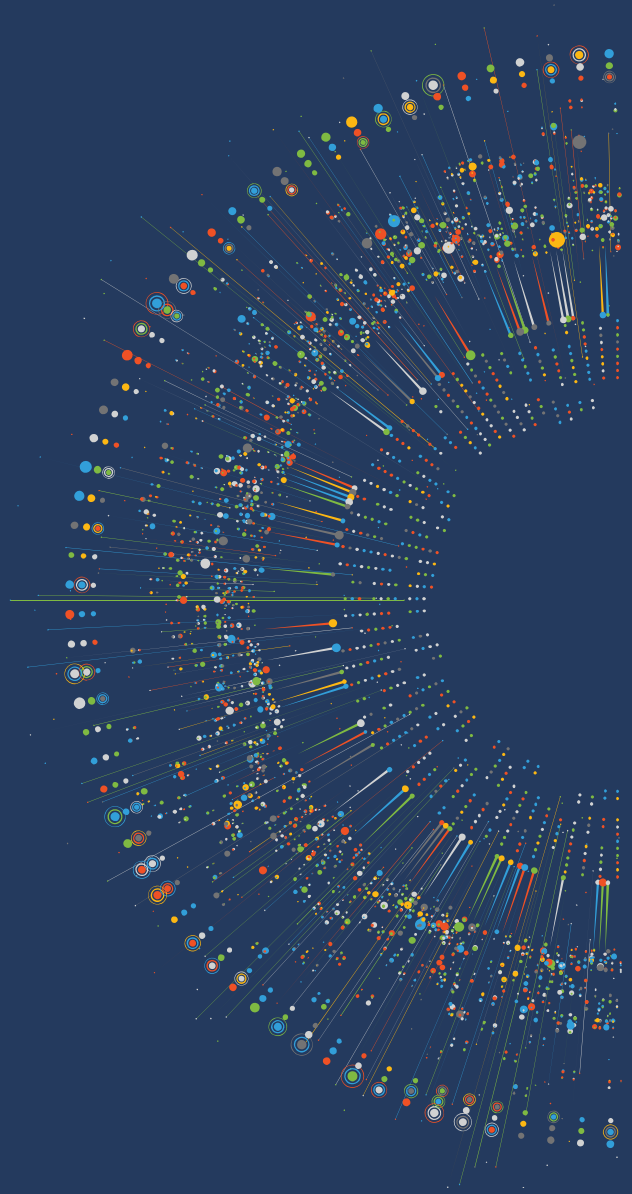


# Azure COMMUNITY CONFERENCE

24th-26th November, 2020

Sponsored by Microsoft

# Speaker

## EDWIN VAN WIJK

Principal Solution Architect

InfoSupport Solid Innovator

Microsoft Most Valuable Professional

# Speaker

## SANDER MOLENKAMP

Principal Cloud Architect

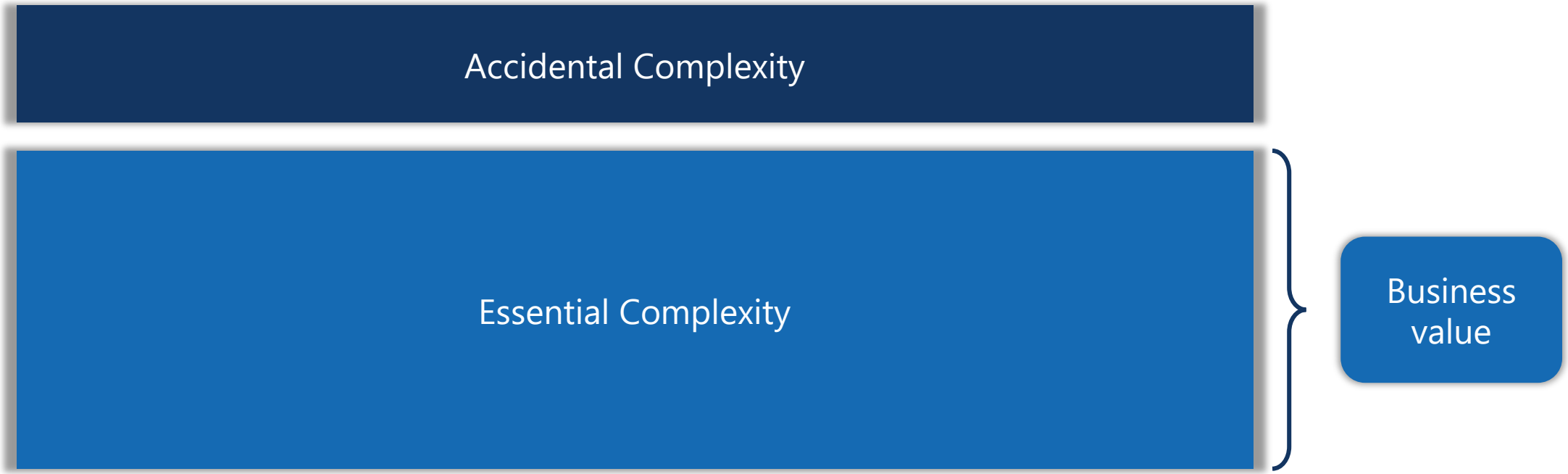
InfoSupport Solid Innovator

MVP Microsoft Most Valuable Professional

# eShop On Dapr

Build microservices the easy way

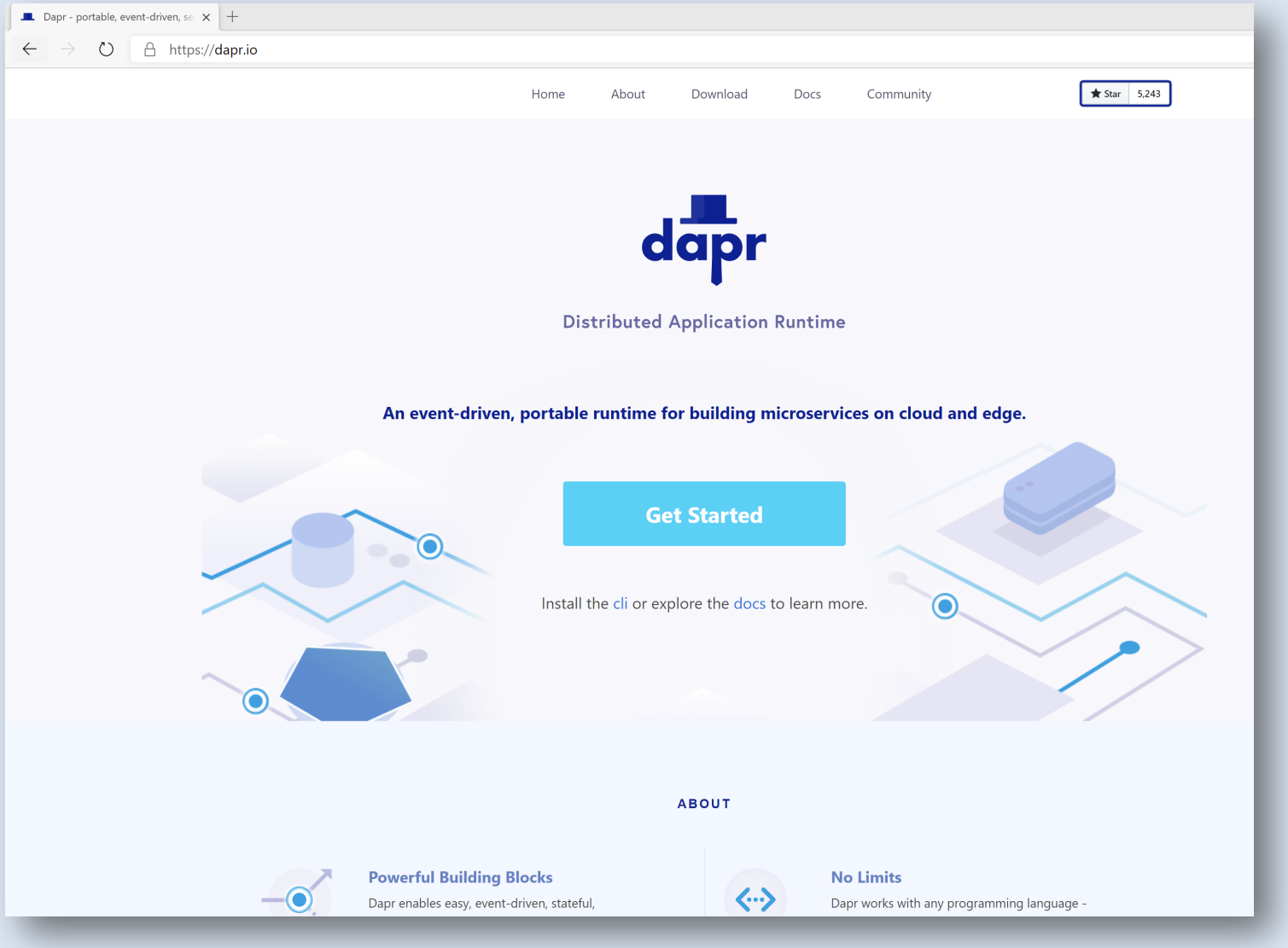
# Complexity


Accidental Complexity
Essential Complexity
Business value

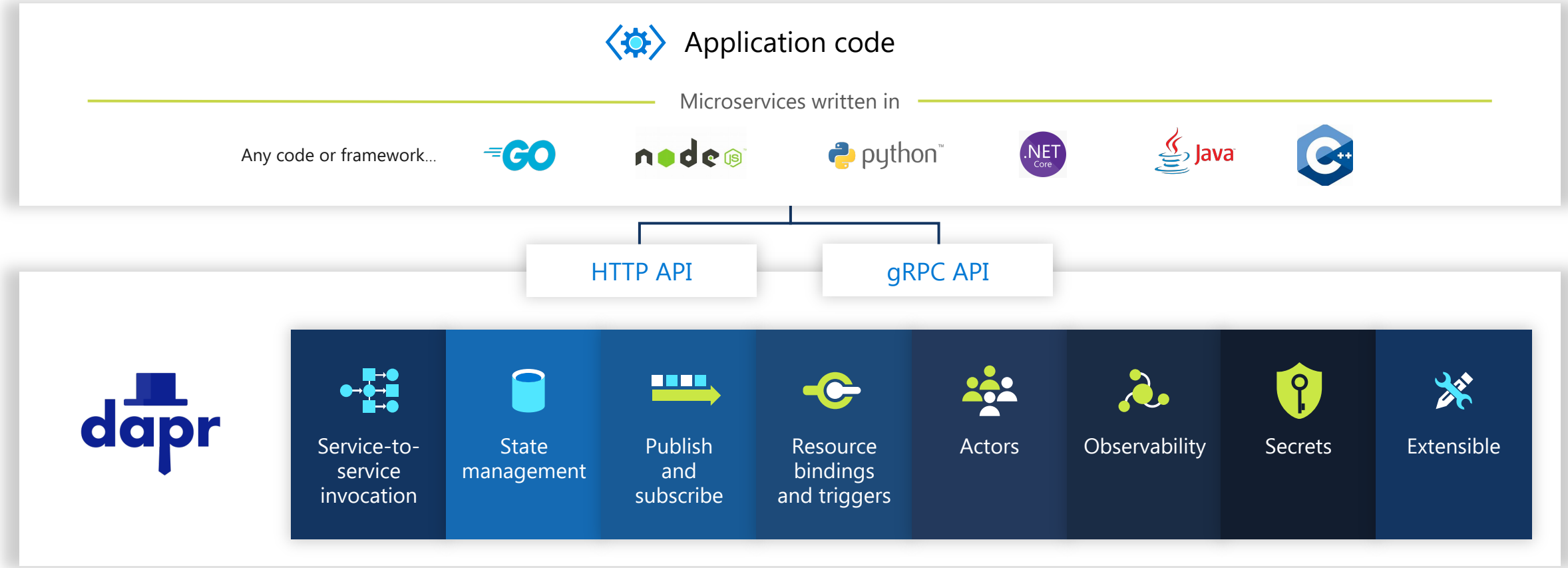
# Distributed Application Runtime

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

https://dapr.io

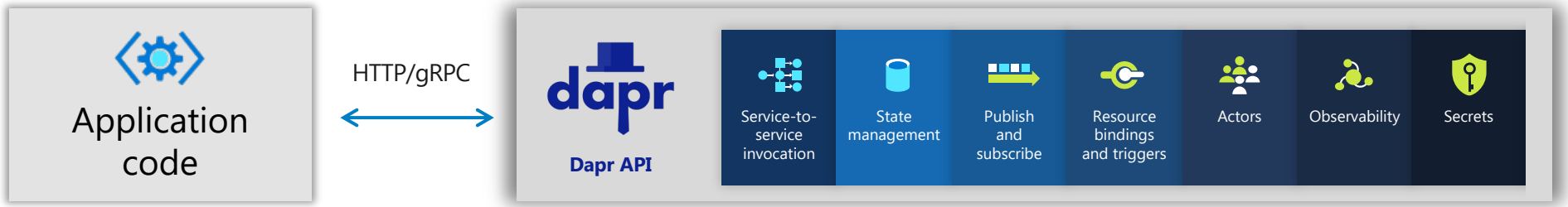


# Microservice building blocks

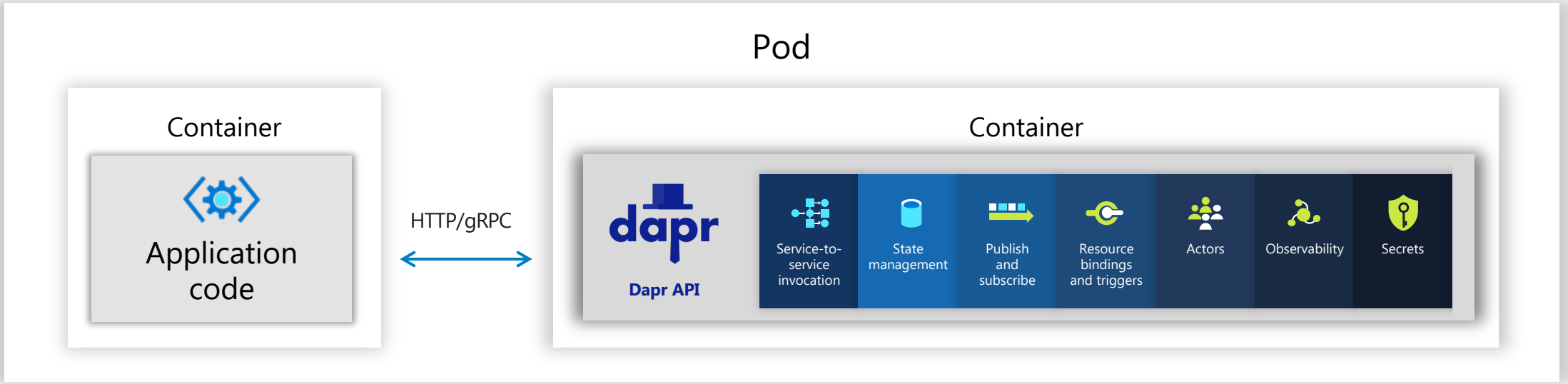


# Sidecar architecture

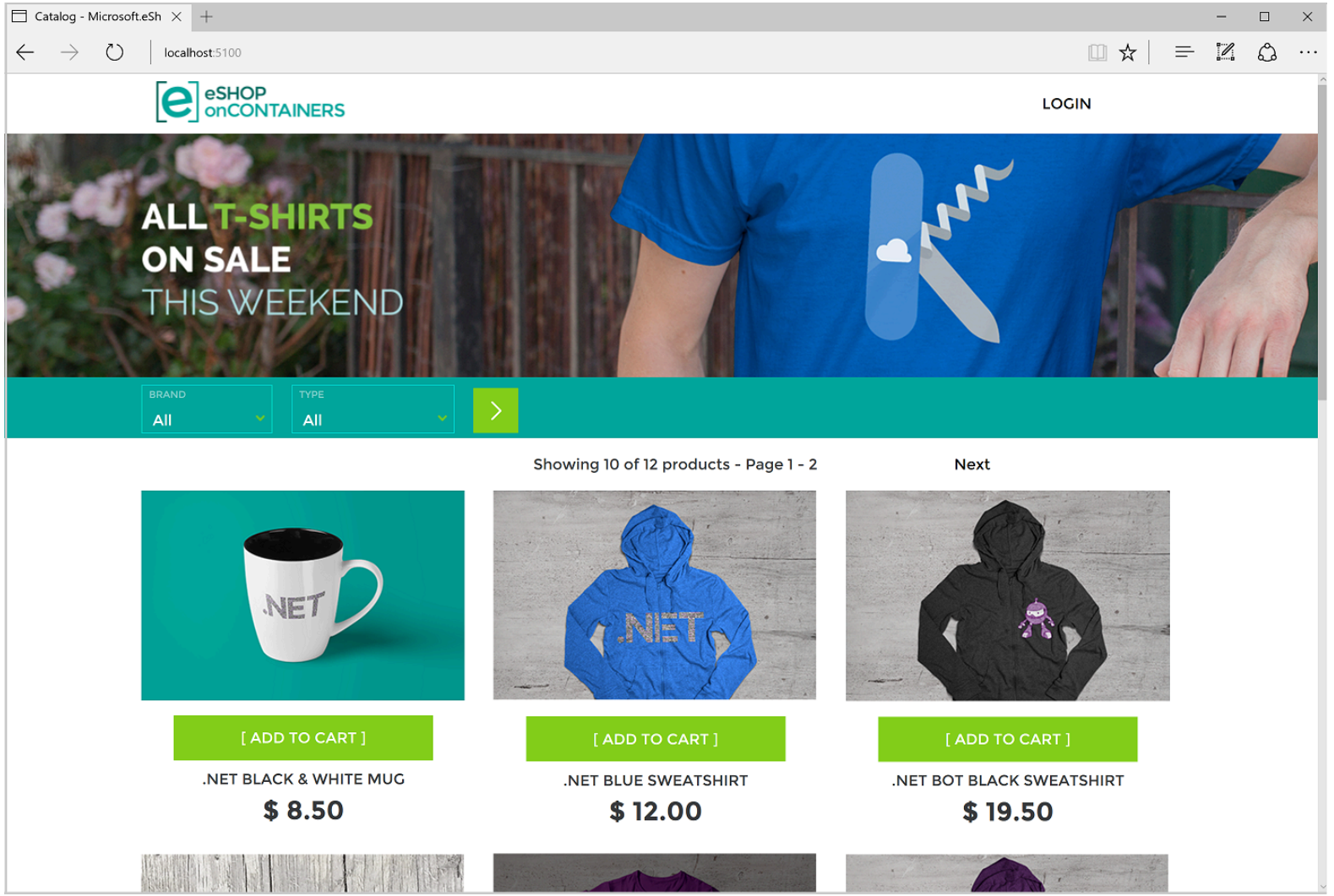
## Standalone



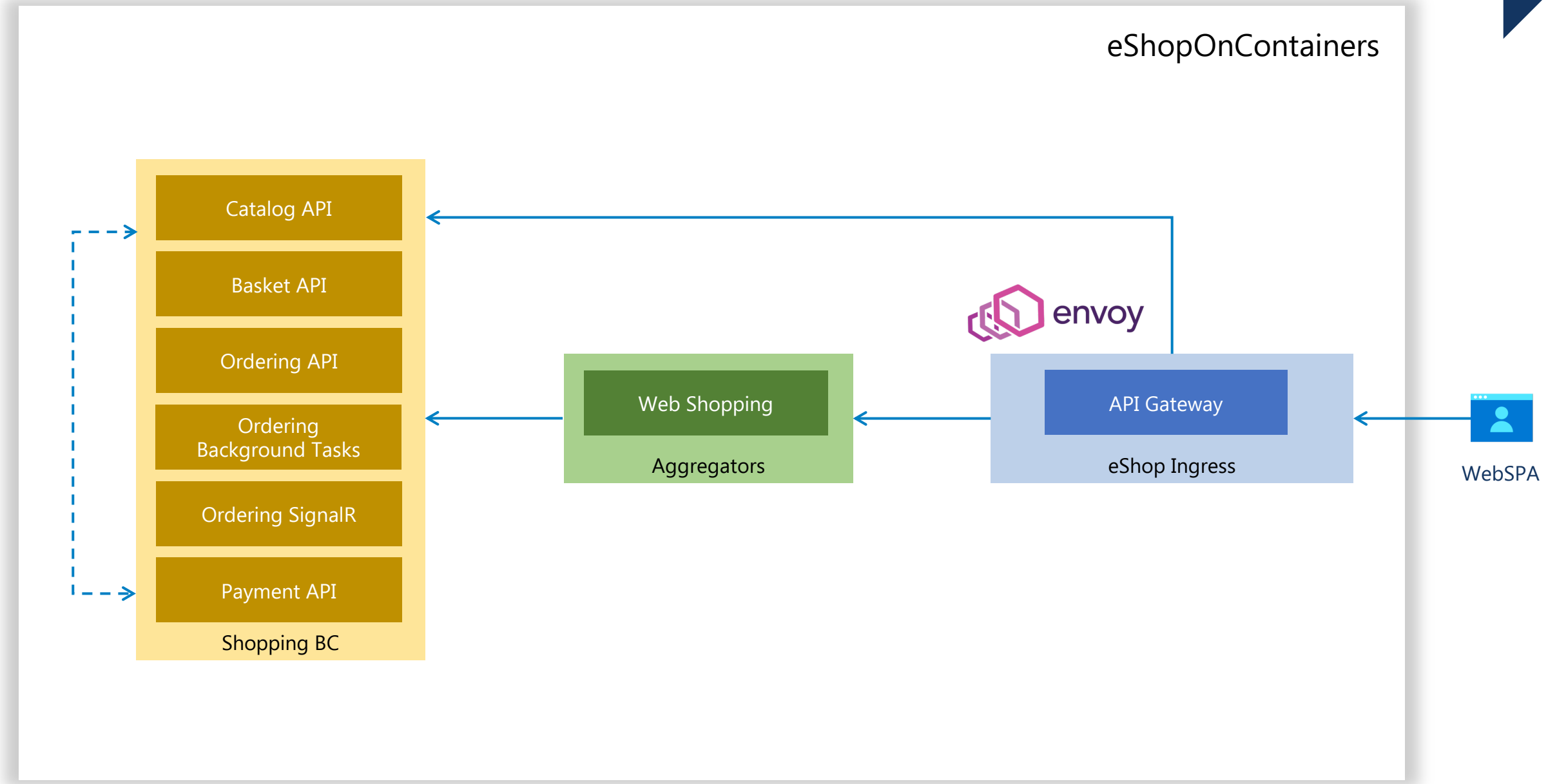

## Kubernetes

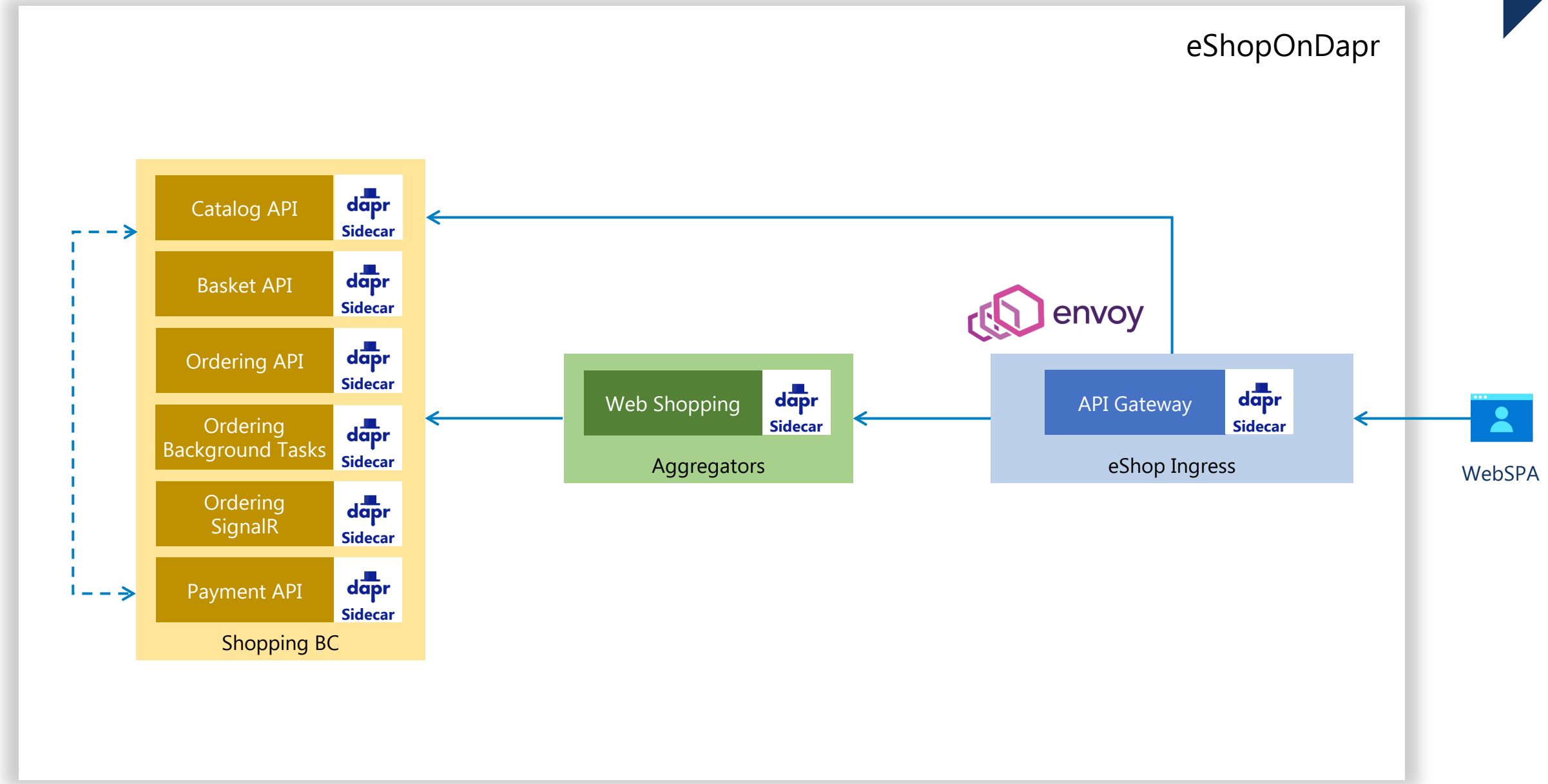


# eShopOnContainers




https://github.com/dotnet-architecture/eShopOnContainers


eShopOnContainers
Catalog API
Basket API
Ordering API
Ordering Background Tasks
Ordering SignalR
Payment API
Shopping BC
Web Shopping
Aggregators
envoy
API Gateway
eShop Ingress
WebSPA


eShopOnDapr
Catalog API
dapr Sidecar
Basket API
dapr Sidecar
Ordering API
dapr Sidecar
Ordering Background Tasks
dapr Sidecar
Ordering SignalR
dapr Sidecar
Payment API
dapr Sidecar
Shopping BC
envoy
Web Shopping
dapr Sidecar
Aggregators
API Gateway
dapr Sidecar
eShop Ingress
WebSPA


# eShopOnDapr

# Microservice 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

## Resource bindings and triggers

Trigger code through events from a large array of inputs

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

# Microservice 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



## Resource bindings and triggers

Trigger code through events from a large array of inputs

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

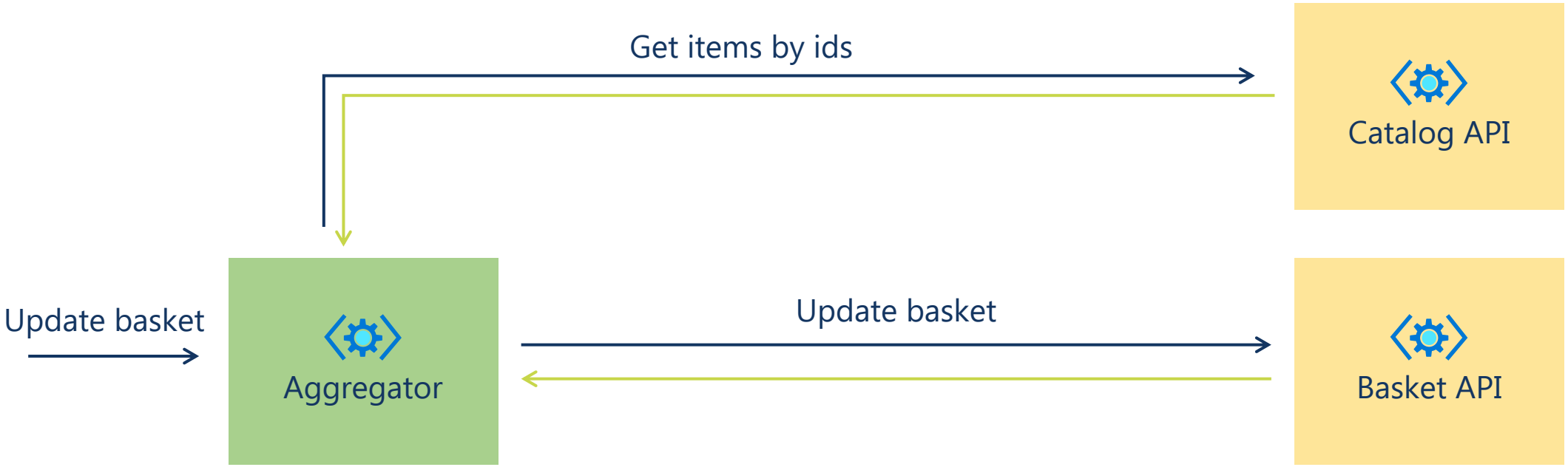
See and measure the message calls across components and networked services



## Secrets

Securely access secrets from your application

# Service calls in eShopOnContainers



# Service calls in eShopOnContainers

# Service calls in eShopOnContainers


```
public class BasketService : IBasketService
{
    private readonly UrlsConfig _urls;
    public readonly HttpClient _httpClient;
    private readonly ILogger<BasketService> _logger;

    …

    public async Task UpdateAsync(BasketData currentBasket)
    {
        await GrpcCallerService.CallService(_urls.GrpcBasket, async channel =>
        {
            var client = new Basket.BasketClient(channel);
            _logger.LogDebug("Grpc update basket currentBasket {@currentBasket}", currentBasket);
            var request = MapToCustomerBasketRequest(currentBasket);
            _logger.LogDebug("Grpc update basket request {@request}", request);

            return await client.UpdateBasketAsync(request);
        });
    }
```

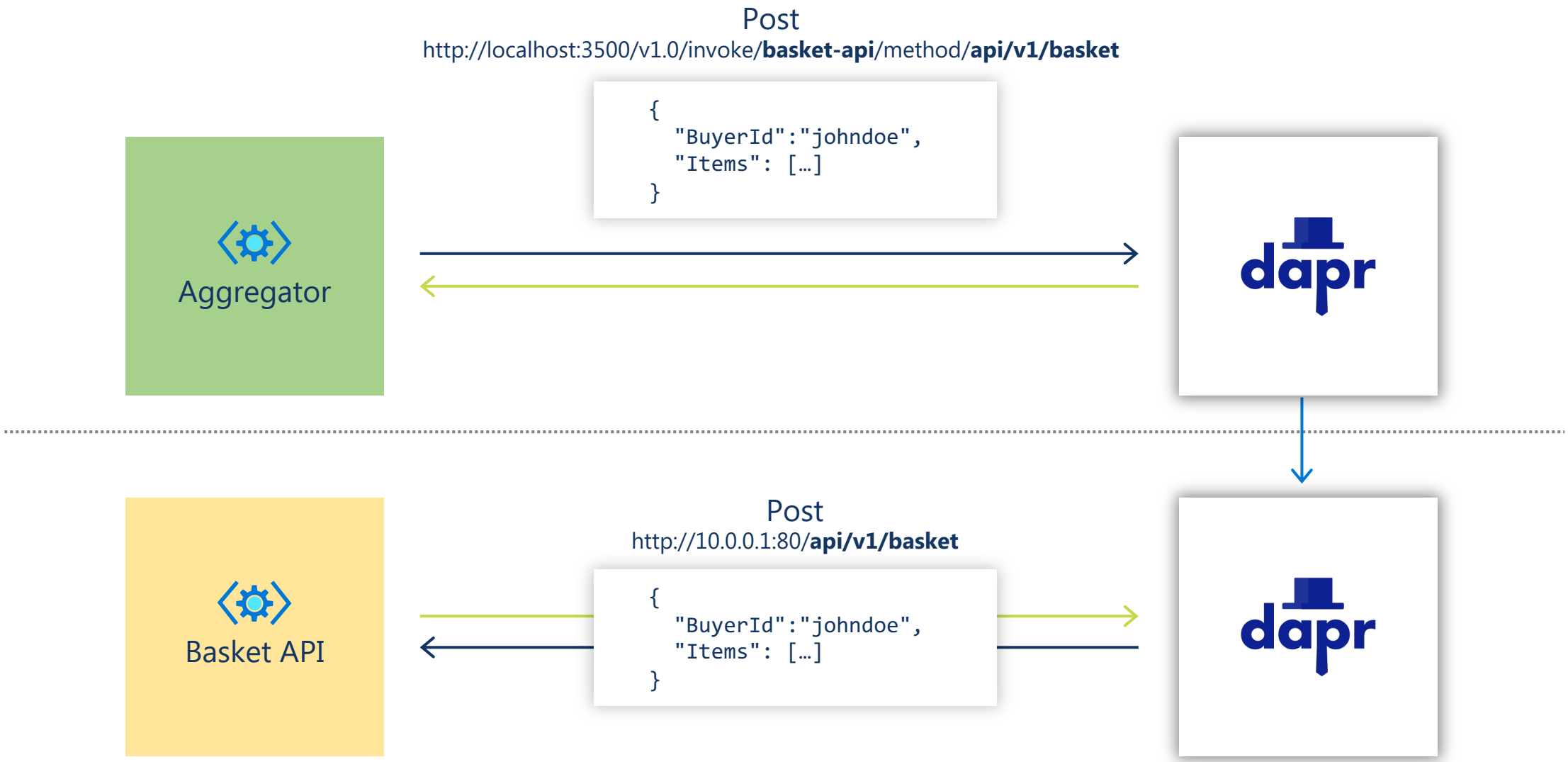
# Service calls in eShopOnContainers


*docker-compose.override.yml*

```
webshoppingagg:
    environment:
      - ASPNETCORE_ENVIRONMENT=Development
      - urls__grpcBasket=http://basket-api:81
      - urls__grpcCatalog=http://catalog-api:81
      - urls__grpcOrders=http://ordering-api:81
      - …
```

```
public class BasketService : IBasketService
{
    private readonly UrlsConfig _urls;
    public readonly HttpClient _httpClient;
    private readonly ILogger<BasketService> _

    …

    public async Task UpdateAsync(BasketData currentBasket)
    {
        await GrpcCallerService.CallService(_urls.GrpcBasket, async channel =>
        {
            var client = new Basket.BasketClient(channel);
            _logger.LogDebug("Grpc update basket currentBasket {@currentBasket}", currentBasket);
            var request = MapToCustomerBasketRequest(currentBasket);
            _logger.LogDebug("Grpc update basket request {@request}", request);

            return await client.UpdateBasketAsync(request);
        });
    }
```

# Service-to-service invocation



# Service calls with Dapr .NET SDK


```proto
service AppCallback {

  rpc OnInvoke (common.v1.InvokeRequest) returns (common.v1.InvokeResponse) {}

  rpc ListTopicSubscriptions(google.protobuf.Empty) returns (ListTopicSubscriptionsResponse) {}

  rpc OnTopicEvent(TopicEventRequest) returns (google.protobuf.Empty) {}

  rpc ListInputBindings(google.protobuf.Empty) returns (ListInputBindingsResponse) {}

  rpc OnBindingEvent(BindingEventRequest) returns (BindingEventResponse) {}
}
```


# Service to service invocation

# Service invocation benefits

# Microservice 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



## Resource bindings and triggers

Trigger code through events from a large array of inputs

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

# Microservice 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



## Resource bindings and triggers

Trigger code through events from a large array of inputs

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

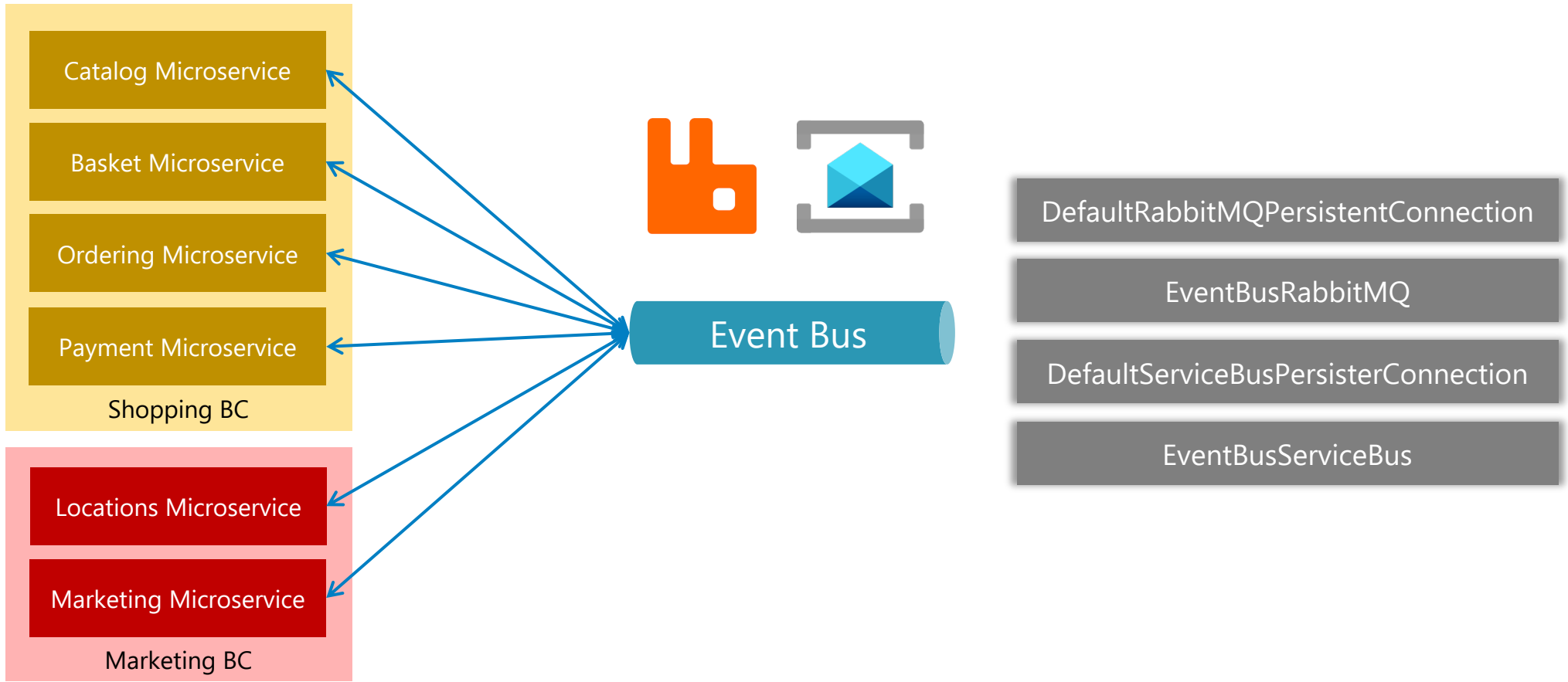
See and measure the message calls across components and networked services



## Secrets

Securely access secrets from your application

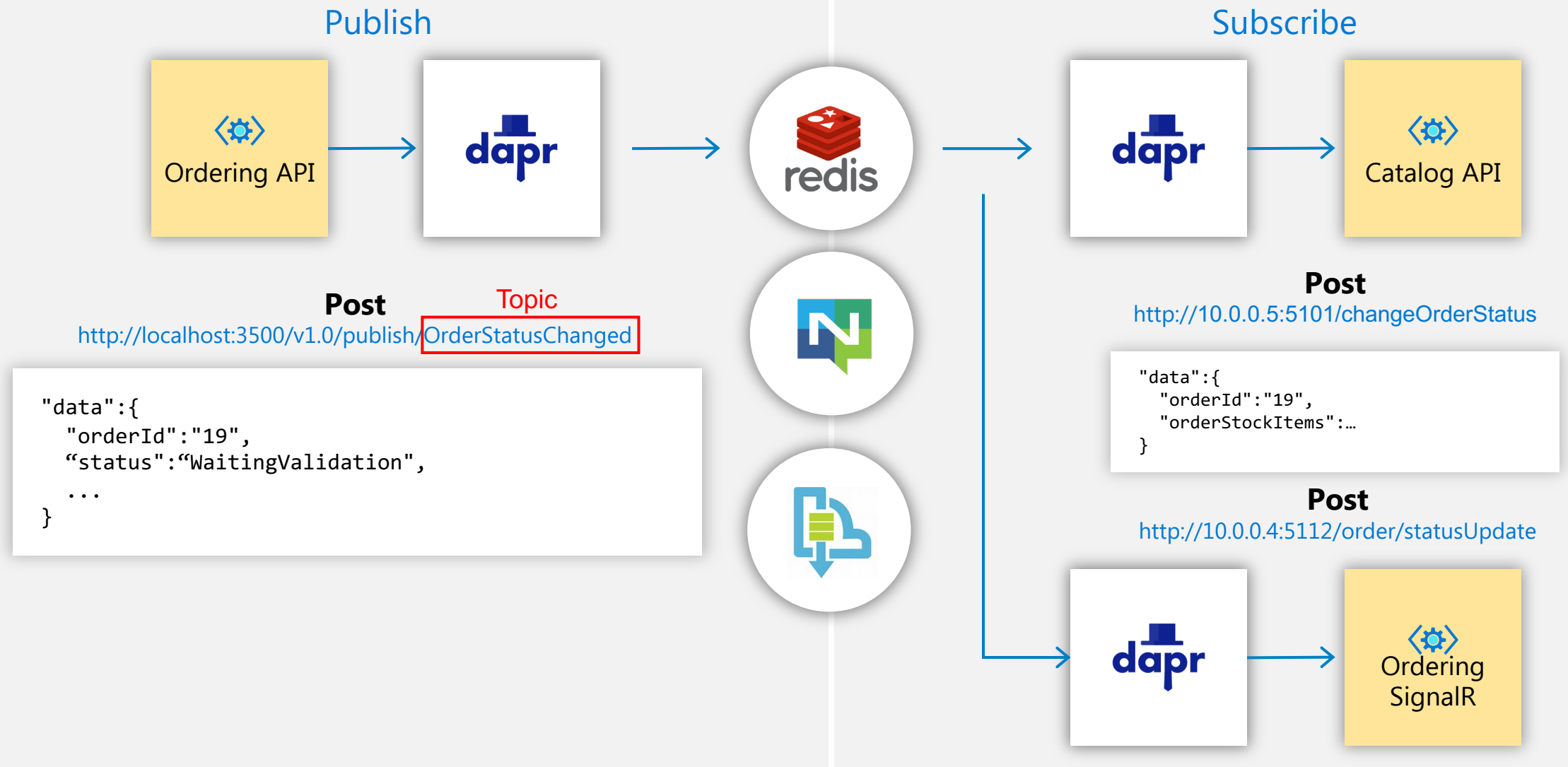
# Publish/Subscribe in eShopOnContainers



# Publish/Subscribe in eShopOnContainers

```
if (configuration.GetValue<bool>("AzureServiceBusEnabled"))
{
    services.AddSingleton<IEventBus, EventBusServiceBus>(sp =>
    {
        var serviceBusPersisterConnection = sp.GetRequiredService<IServiceBusPersisterConnection>();
        var iLifetimeScope = sp.GetRequiredService<ILifetimeScope>();
        var logger = sp.GetRequiredService<ILogger<EventBusServiceBus>>();
        var eventBusSubcriptionsManager = sp.GetRequiredService<IEventBusSubscriptionsManager>();

        return new EventBusServiceBus(serviceBusPersisterConnection, logger,
            eventBusSubcriptionsManager, subscriptionClientName, iLifetimeScope);
    });
}
else
{
    services.AddSingleton<IEventBus, EventBusRabbitMQ>(sp =>
    {
        var rabbitMQPersistentConnection = sp.GetRequiredService<IRabbitMQPersistentConnection>();
        var iLifetimeScope = sp.GetRequiredService<ILifetimeScope>();
        var logger = sp.GetRequiredService<ILogger<EventBusRabbitMQ>>();
        var eventBusSubcriptionsManager = sp.GetRequiredService<IEventBusSubscriptionsManager>();

        var retryCount = 5;
        if (!string.IsNullOrEmpty(configuration["EventBusRetryCount"]))
        {
            retryCount = int.Parse(configuration["EventBusRetryCount"]);
        }

        return new EventBusRabbitMQ(rabbitMQPersistentConnection, logger, iLifetimeScope, eventBusSubcriptionsManager, subscriptionClientName, retryCount);
    });
}
```

# Publish/Subscribe



# ASP.NET Core Integration

Topic

```
[Topic("pubsub", "OrderStatusChanged")]
[HttpPost("/changeOrderStatus")]
public async Task<ActionResult> StatusChange(OrdertatusChange statusChange)
{
    ...
}
```

Topic

```
[Topic("pubsub", "OrderStatusChanged")]
[HttpPost("/order/statusUpdate")]
public async Task<ActionResult> StatusUpdate(StatusChange sc)
{
    ...
}
```

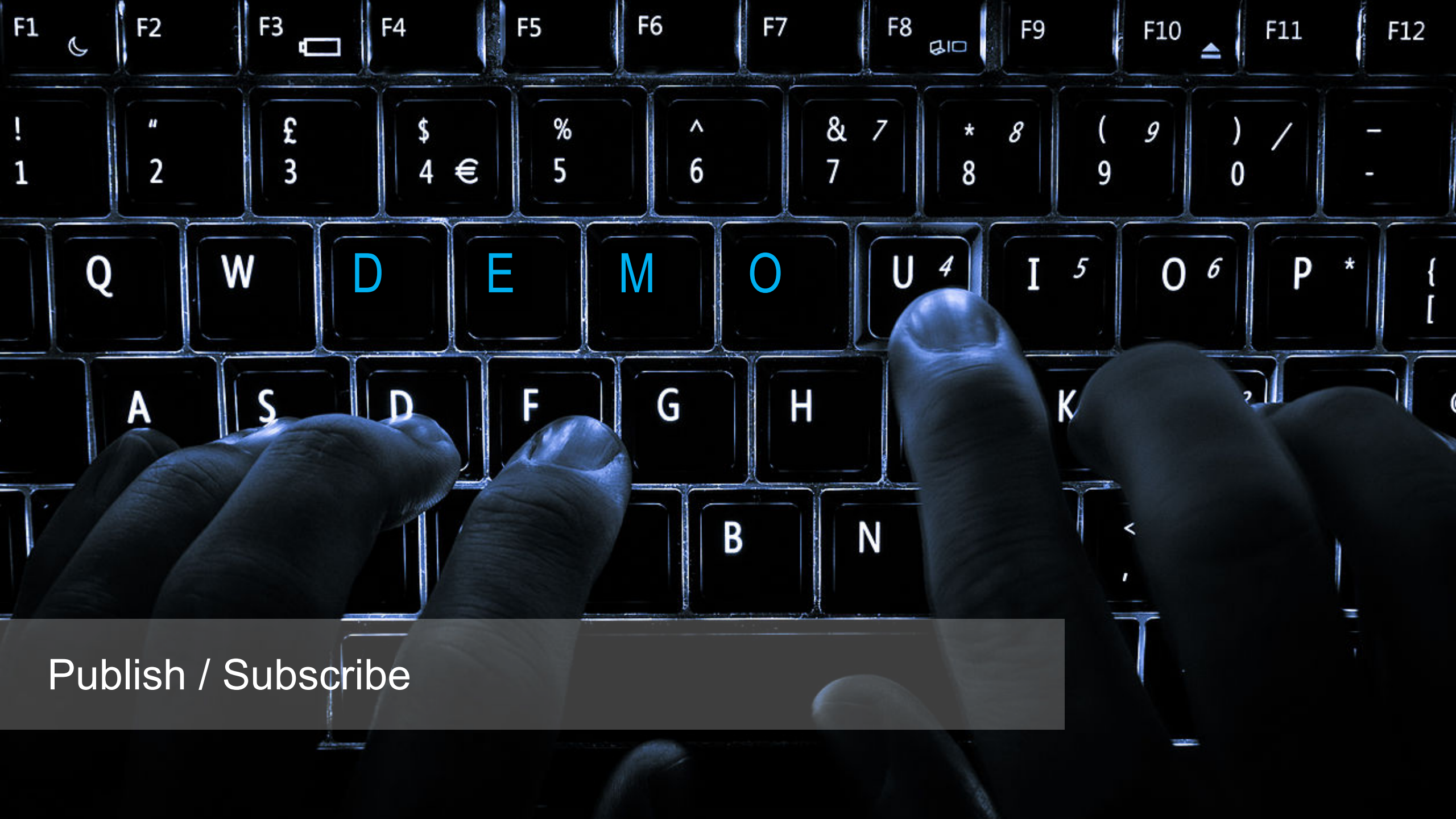
# ASP.NET Core Integration

```
public void ConfigureServices(IServiceCollection services)
{
    ...

    services.AddControllers().AddDapr();
}


public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    ...

    app.UseCloudEvents();

    app.UseEndpoints(endpoints =>
    {
        endpoints.MapSubscribeHandler();

        ...
    });
}
```

# Component Configuration

- Component are configured using configuration files
- Name of the component can be specified in code
- Each component has a specific type
- Each type has specific metadata

```yaml
apiVersion: dapr.io/v1alpha1
kind: Component
metadata:
  name: pubsub
  namespace: eshop
spec:
  type: pubsub.natsstreaming
  metadata:
  - name: natsURL
    value: nats://stan:4222
  - name: natsStreamingClusterID
    value: eShopOnDapr
  - name: subscriptionType
    value: queue
```




# Publish / Subscribe

# Microservice 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



## Resource bindings and triggers

Trigger code through events from a large array of inputs

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

# Microservice 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



## Resource bindings and triggers

Trigger code through events from a large array of inputs

Output bindings to external resources including databases and queues



## Actors

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



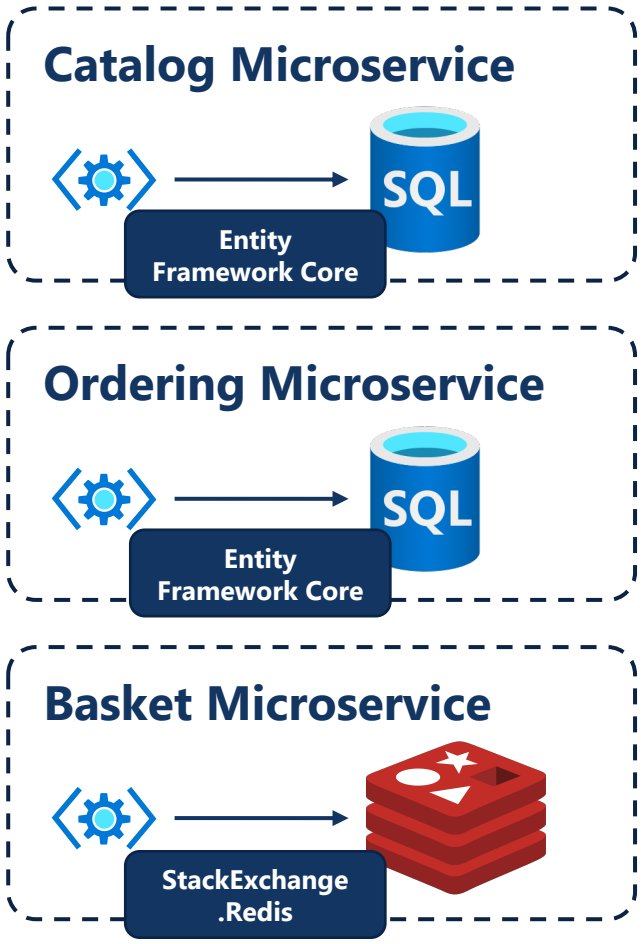
## Observability

See and measure the message calls across components and networked services

## Secrets

Securely access secrets from your application

# State management in eShopOnContainers

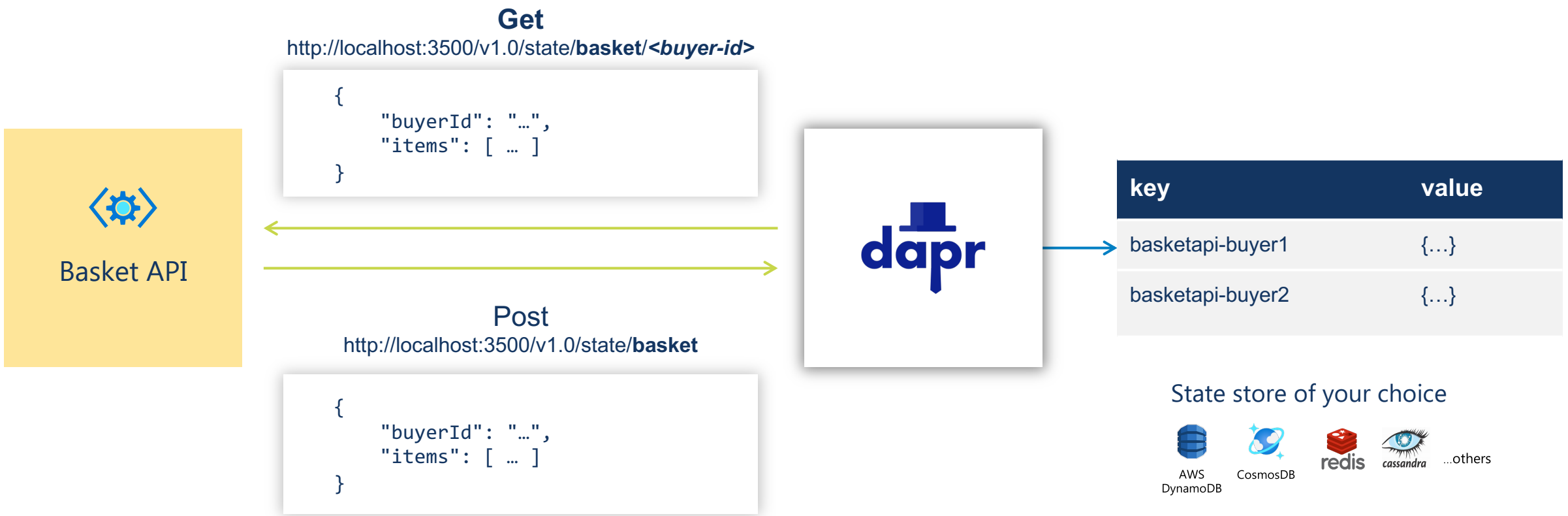


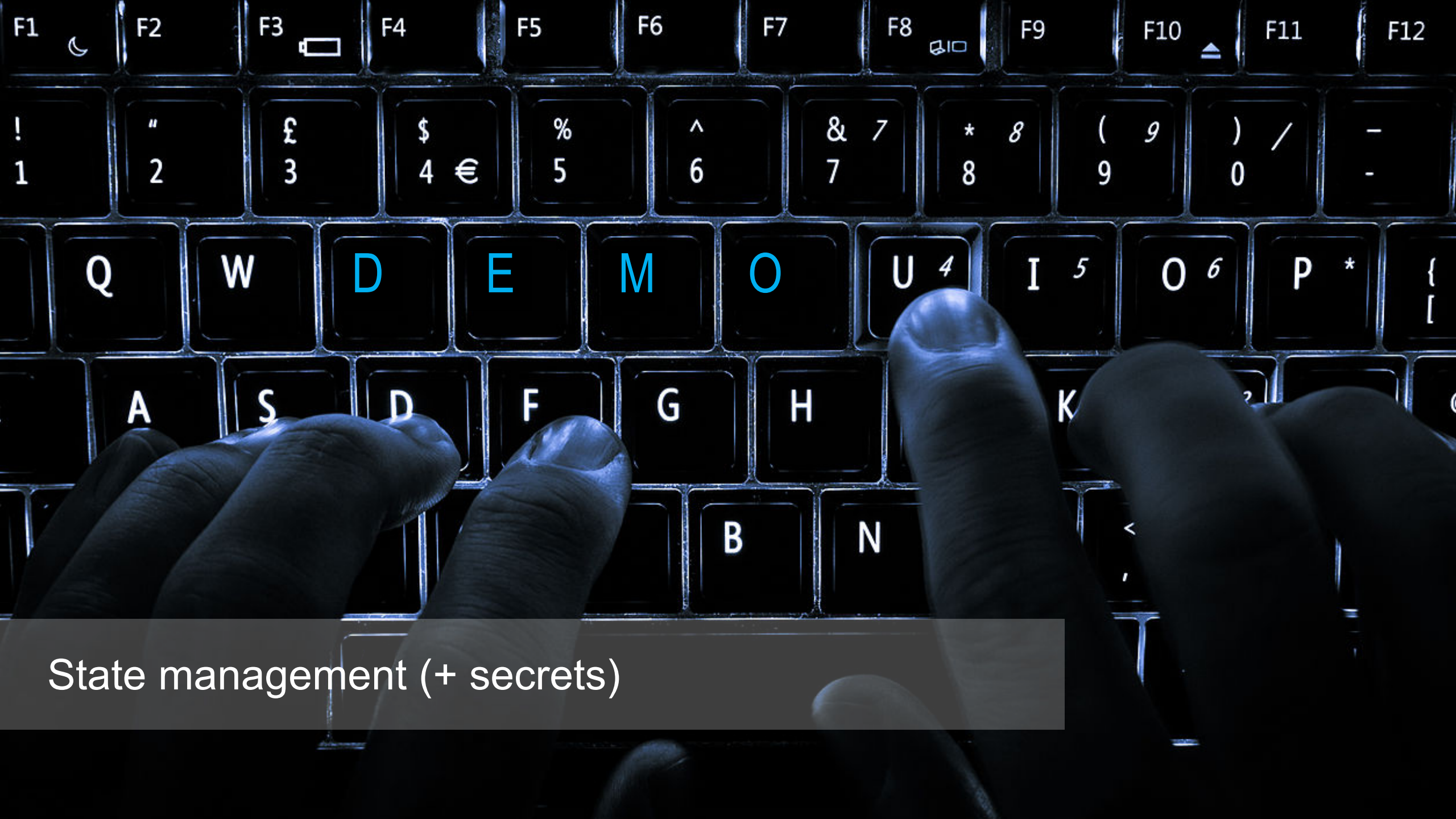

Example of a simple Data-Driven and CRUD microservice using EF Core

Example of a Domain Driven Design Microservice using DDD patterns

Example of a simple Data-Driven and CRUD microservice using Redis

# State management






State management (+ secrets)

# Supported state stores

| Name | CRUD | Transactional |
|---|---|---|
| Aerospike | ✅ | ❌ |
| Cassandra | ✅ | ❌ |
| Cloudstate | ✅ | ❌ |
| Couchbase | ✅ | ❌ |
| etcd | ✅ | ❌ |
| Hashicorp Consul | ✅ | ❌ |
| Hazelcast | ✅ | ❌ |
| Memcached | ✅ | ❌ |
| MongoDB | ✅ | ✅ |
| Redis | ✅ | ✅ |
| Zookeeper | ✅ | ❌ |
| Azure CosmosDB | ✅ | ❌ |
| Azure SQL Server | ✅ | ✅ |
| Azure Table Storage | ✅ | ❌ |
| Google Cloud Firestore | ✅ | ❌ |

# Microservice 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



## Resource bindings and triggers

Trigger code through events from a large array of inputs

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

# Microservice 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

## Resource bindings and triggers

Trigger code through events from a large array of inputs

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

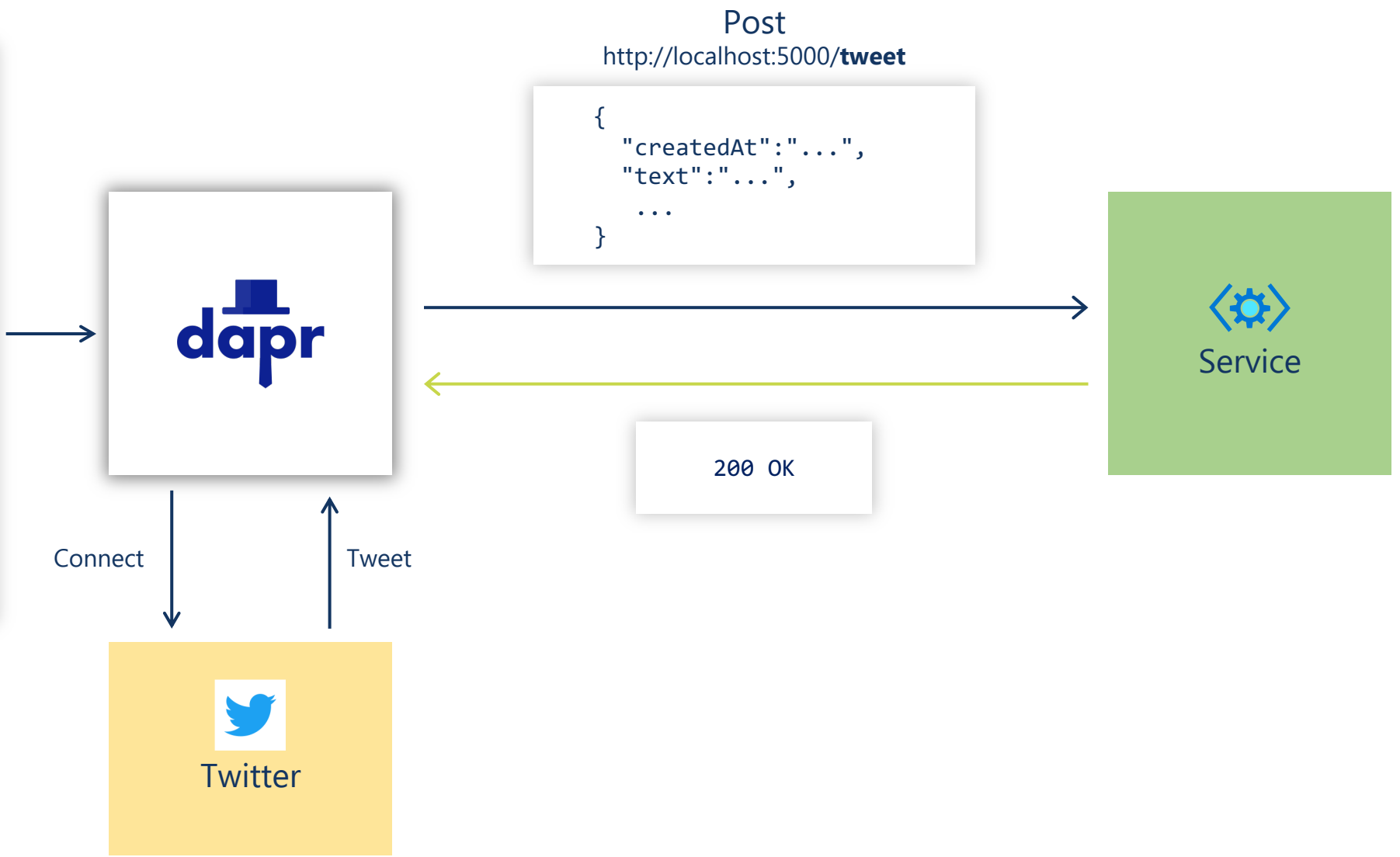
Securely access secrets from your application

# Resource Bindings

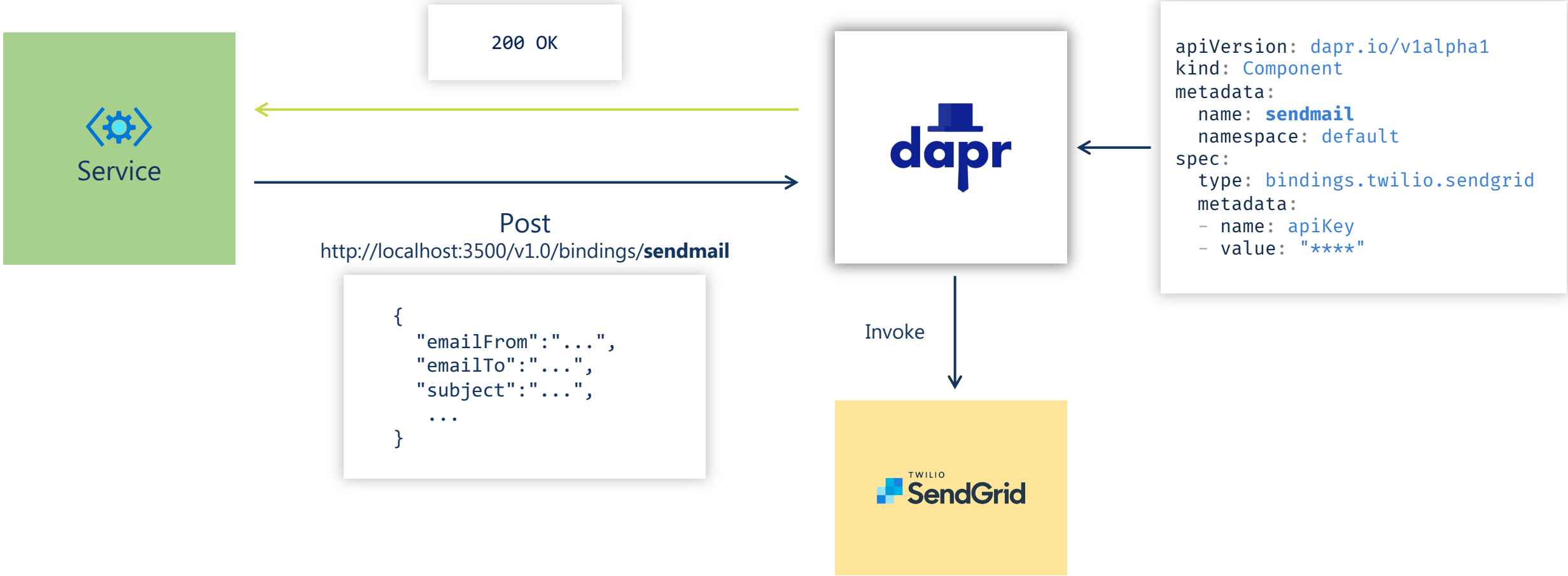
- A way to integrate with different external systems
- Not only message brokers (e.g. Twitter, Twilio SendGrid, Azure Storage)
- Input bindings (or triggers) react to events in the external system and invoke your application
- Output bindings are called from your application (using HTTP or gRPC) and send a message to the external system

# Input Binding (trigger)

```yaml
apiVersion: dapr.io/v1alpha1
kind: Component
metadata:
  name: tweet
  namespace: default
spec:
  type: bindings.twitter
  metadata:
  - name: consumerKey
    value: "****"
  - name: consumerSecret
    value: "****"
  - name: accessToken
    value: "****"
  - name: accessSecret
    value: "****"
  - name: query
    value: "dapr"
```



# Output Binding



# Wrap up / take aways

Reduces amount of code that needs to be written manually

Greatly increases portability and interoperability

Not production ready yet

# Thank you!


Sander

Info Support Solid Innovator

@amolenk


Edwin van Wijk

Info Support Solid Innovator

@evanwijk

# Resources:

All things Dapr:

dapr.io

eShopOnDapr:

github.com/dotnet-architecture/eShopOnDapr

dotnetFlix

www.youtube.com/dotnetflix