Visual Studio LIVE! | AUSTIN
EXPERT SOLUTIONS FOR ENTERPRISE DEVELOPERS

dapr

Workshop
Dapr for building distributed .NET Core applications

Marcel de Vries & Alex Thissen

Cloud architects at Xpirit, The Netherlands

Microsoft® MVP Most Valuable Professional

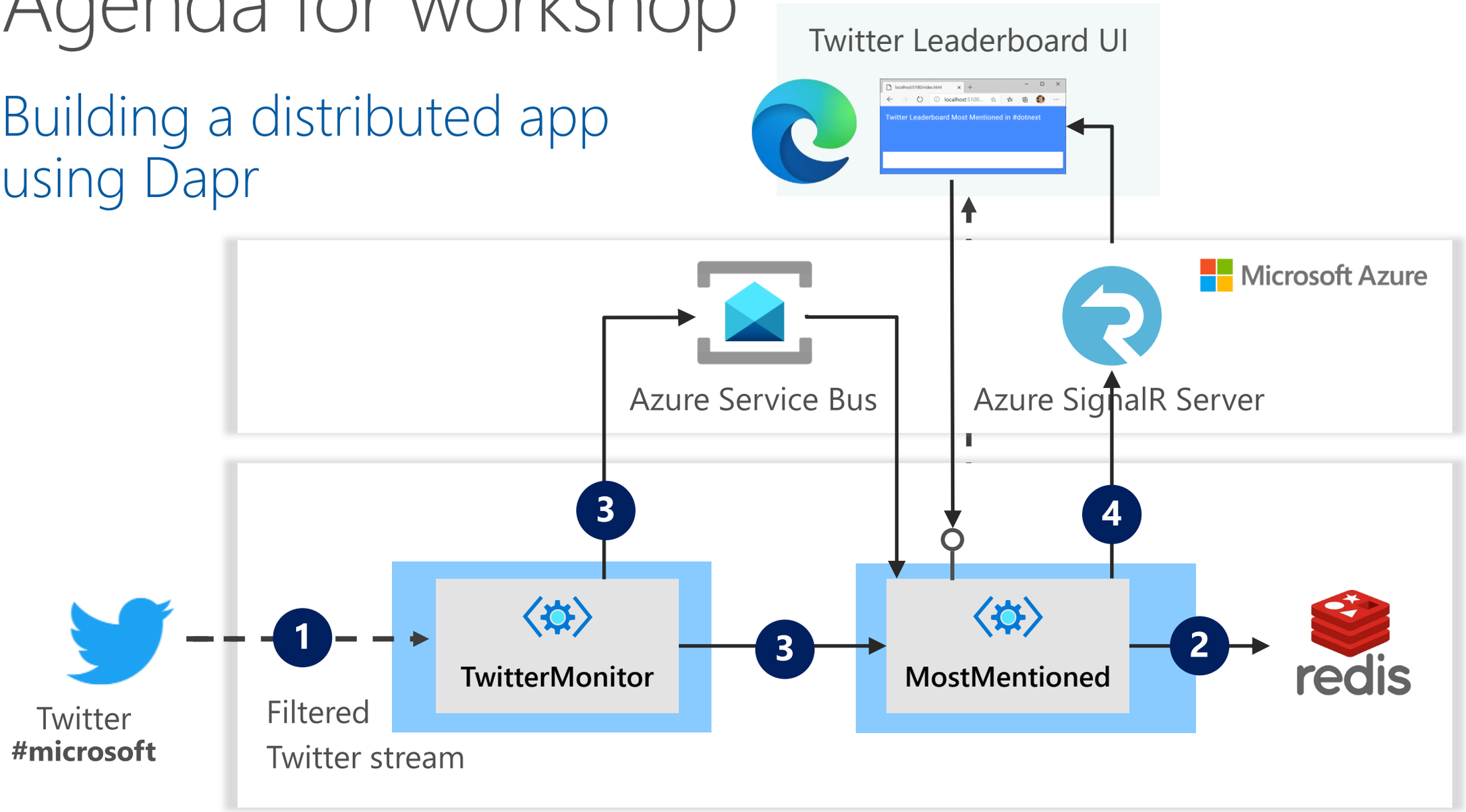
Xpirit

#VSLIVE

NO CODE LIMITS

Agenda for workshop

Building a distributed app using Dapr



Application runtimes & service meshes

Service meshes

Infrastructure-centric (for operators)

Focus on networking concerns

Layer7 routing
Traffic flow management (splitting)
mTLS authentication
Telemetry, metrics and tracing

Not a programming model

Istio
HashiCorp Consul
LINKERD

Distributed application runtime

Programmer-centric

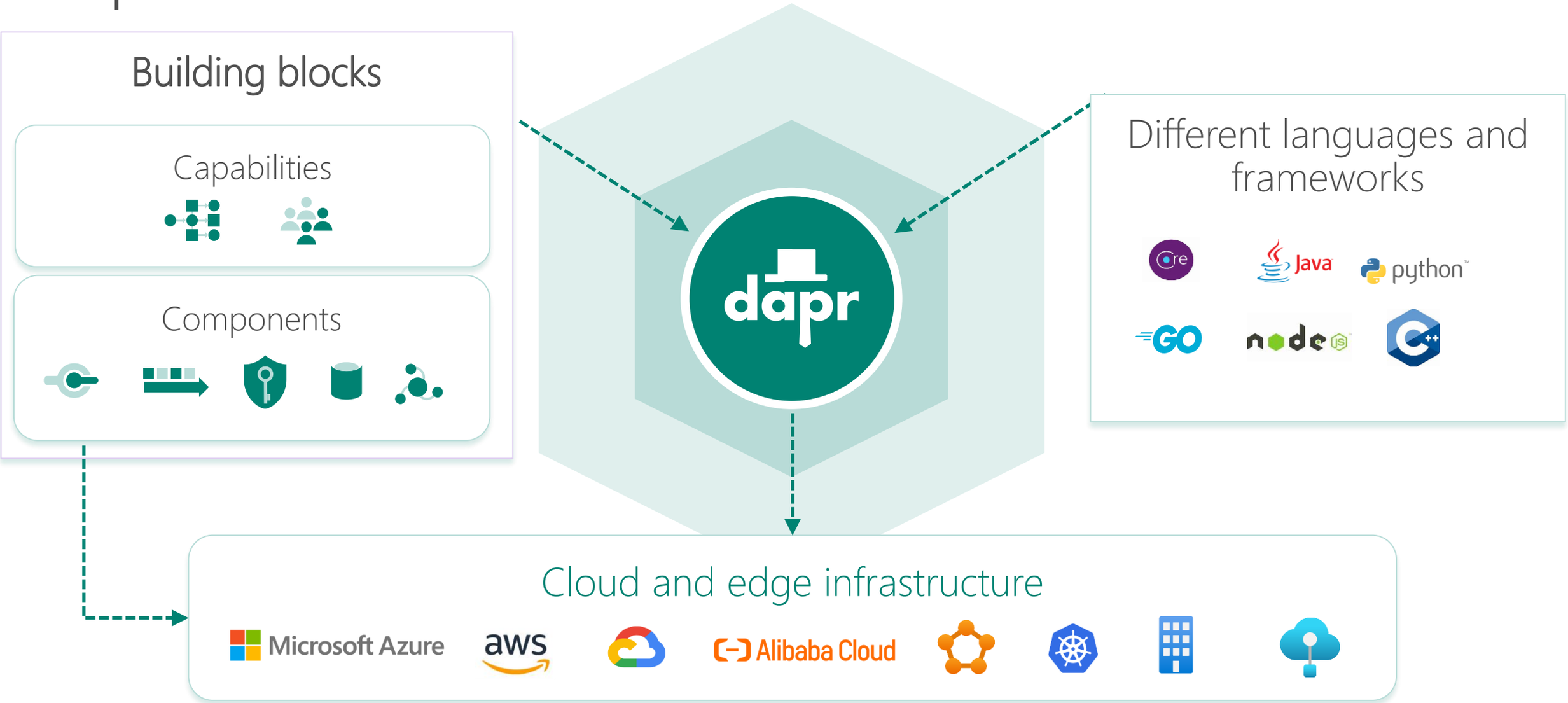
Focuses on building blocks

Provides programming model
Introduces new functionality to app runtime

dapr

*"Dapr is a portable, serverless, event-driven **runtime** that makes it easy for developers to build resilient, stateless and stateful microservices that run on the cloud and edge and embraces the diversity of languages and developer frameworks."*

Dapr overview

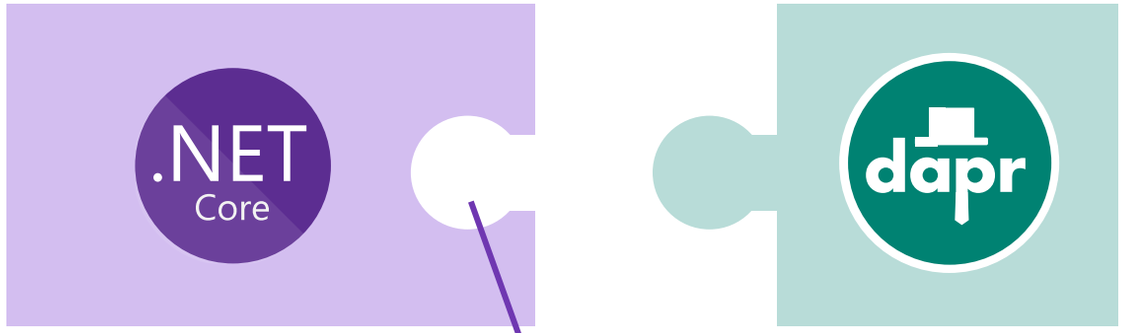


Sidecar architecture

Sidecars attach to application

Not a part of implementation

Provides supporting features and cross-cutting concerns



Application

Sidecar

Non-invasive connection to application

Application must “communicate” with sidecar via sidecar's API (and back)

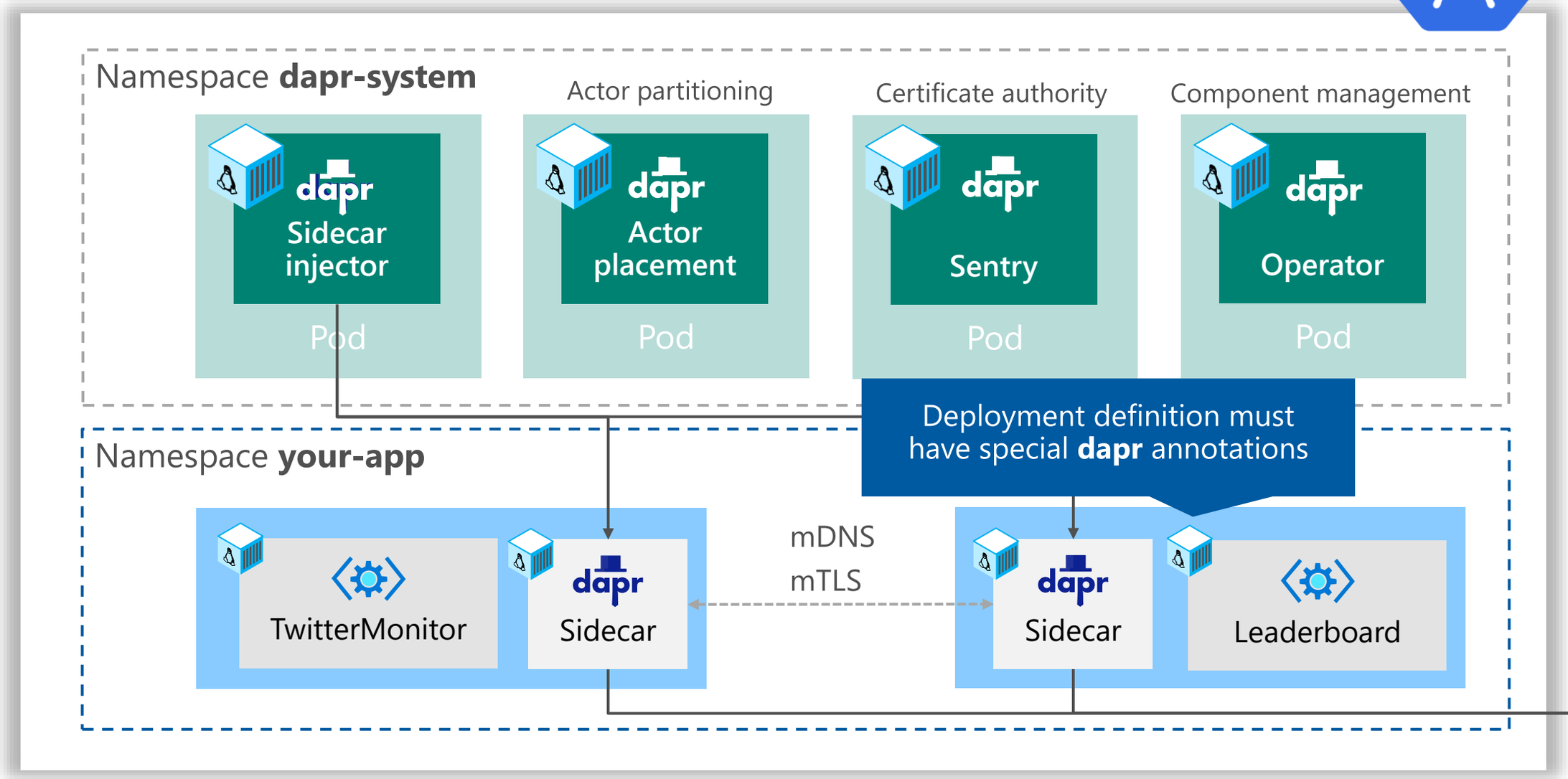
Example scenarios

Infrastructure: health checks, watchdogs, logging, telemetry

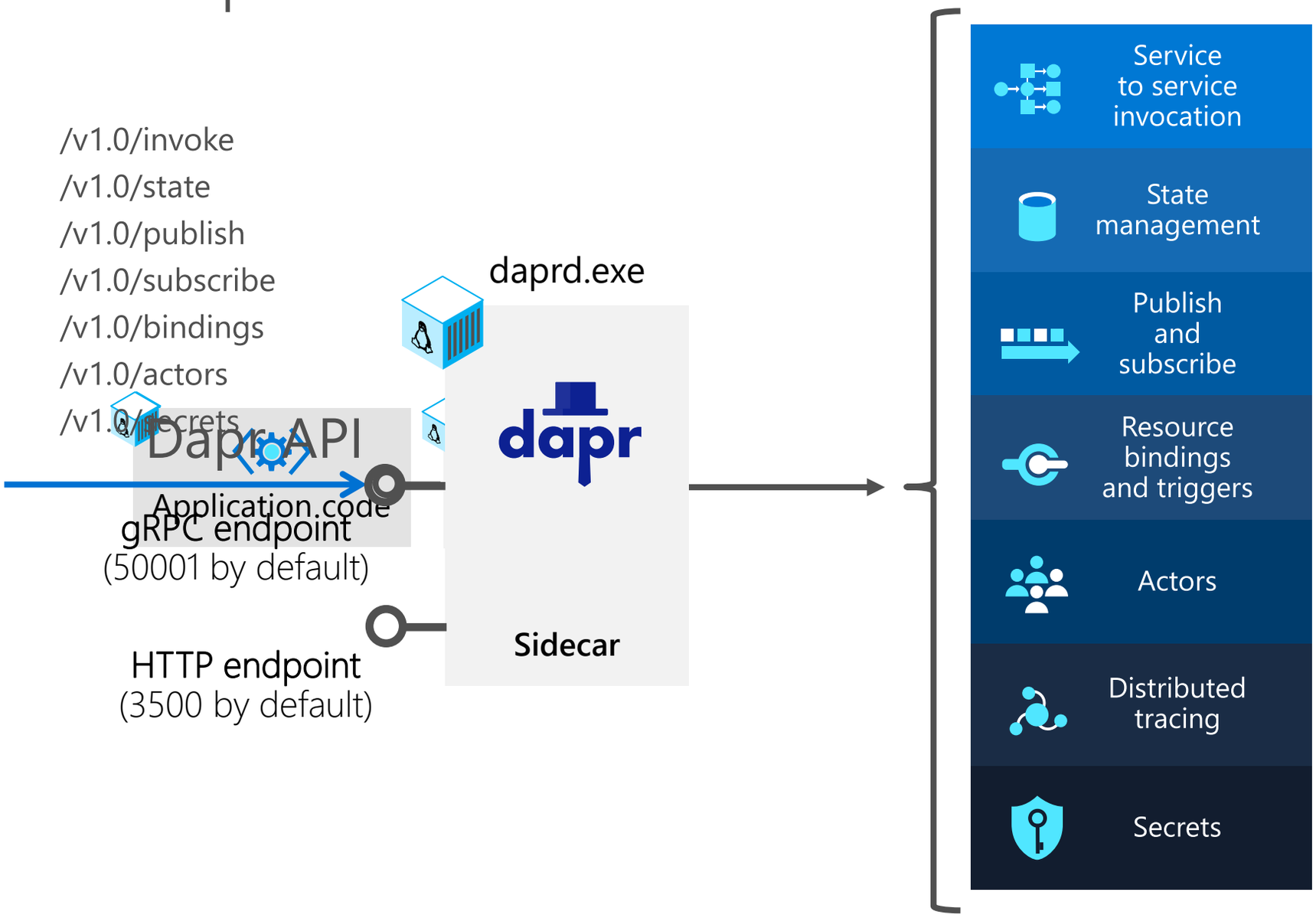
Ambassador: cross-service communication

Offload proxy: handle additional work of application

Running Dapr in Kubernetes



Dapr sidecars up close



Components

Abstraction

Implementation details abstracted away by sidecar

Declarative

Defined in text-based artifacts and outside of application

Requires no binaries

Components

Publish and subscribe	Azure Service Bus and Event Hubs, NATS, SNS SQS, GCP Pub/Sub, Hazelcast, Kafka, MQTT, Pulsar, RabbitMQ, Redis Streams
Resource bindings and triggers	
State Management	Aerospike, AWS DynamoDB, Azure CosmosDB and Table Storage, Cassandra, CloudState, Couchbase, Etcd, GCP Firestore, Hashicorp Consul, Hazelcast, Memcached, MongoDB, Redis Streams, SQL Server, Zookeeper
Distributed tracing	Zipkin, String (testing), Native
Secrets	AWS Secret Manager, Azure KeyVault, GCP Cloud KMS and Secret Manager, Hashicorp Vault, JSON files and Kubernetes

Declarations in Dapr

Components

```yaml
apiVersion: dapr.io/v1alpha1
kind: Component
metadata:
  name: zipkin
  namespace: default
spec:
  type: exporters.zipkin
  metadata:
  - name: enabled
    value: "true"
  - name: exporterAddress
    value: "http://zipkin:941
```

Configuration

```yaml
apiVersion: dapr.io/v1a
kind: Configuration
metadata:
  name: appconfig
  namespace: default
spec:
  tracing:
    samplingRate: "1"
```

Kubernetes annotations

```yaml
apiVersion: apps/v1
kind: Deployment
…
spec:
  template:
    metadata:
      annotations:
        dapr.io/enabled: "true"
        dapr.io/id: "myapp"
        dapr.io/port: "5000"
        dapr.io/config: "appconfig"
    spec:
      containers:
      - name: netcoreapp
        image: leaderboard:latest
```

Getting up and running

1 Install Docker

Comes with a local Kubernetes cluster for development purposes

2 Install Dapr CLI

```
      __
   ____/ /___ _____  _____
  / __  / __ '/ __ \/ ___/
 / /_/ / /_/ / /_/ / /
 \__,_/\__,_/ .___/_/
           /_/
=====================================================
A serverless runtime for hyperscale, distributed systems
```

Local mode (standalone)

List, start and stop dapr instances

Kubernetes only

List components and configurations
Show logs for pod
Status of system services

3 Initialize Dapr runtime

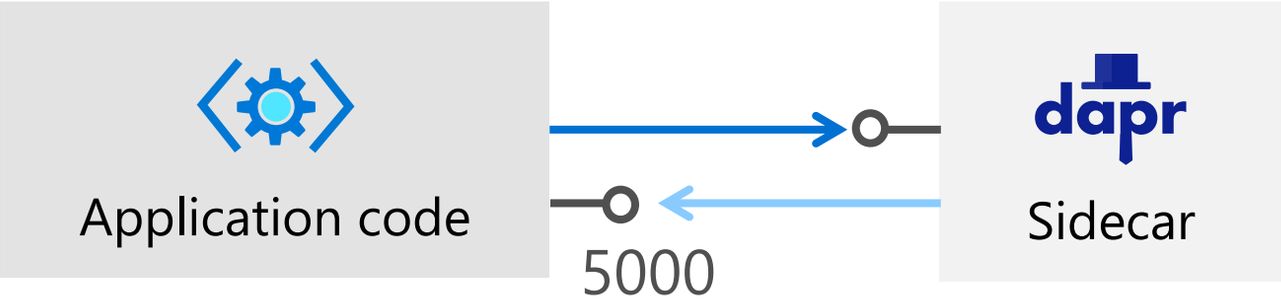
Standalone for local mode during development
Kubernetes in production (with Dapr CLI or Helm 3)

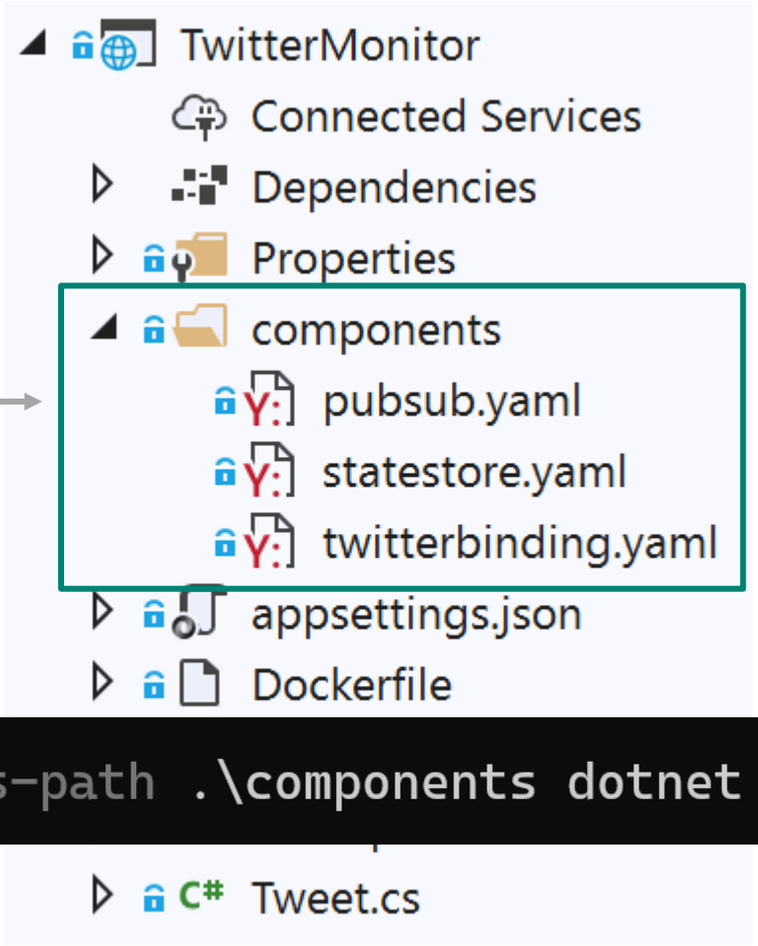
Running with sidecars locally

Dapr supports standalone mode

daprd.exe can attach sidecar to single process

Dapr CLI to start, stop and list instances






```
dapr run --app-id hello --app-port 5000 --components-path .\components dotnet run
```

Dapr sidecar and .NET Core

ASP.NET Core

Builder extensions
State model binding
Cloud Events middleware
Subscription registration and routing
Actor integration

Actors

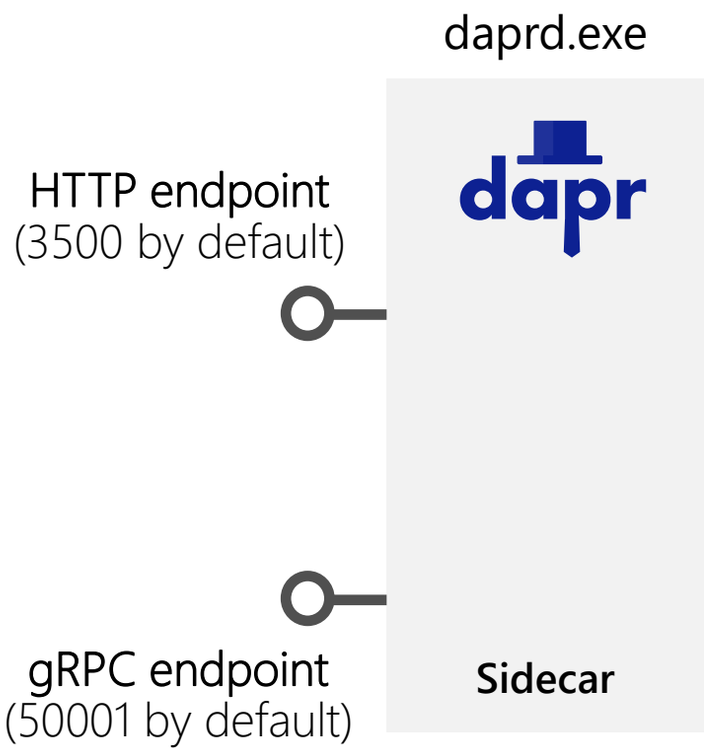
Programming model
Timers and reminders

Configuration

Secret store extensions

Dapr client

Dapr gRPC client (builder and proto)
Concurrency, retry and state options



Capabilities

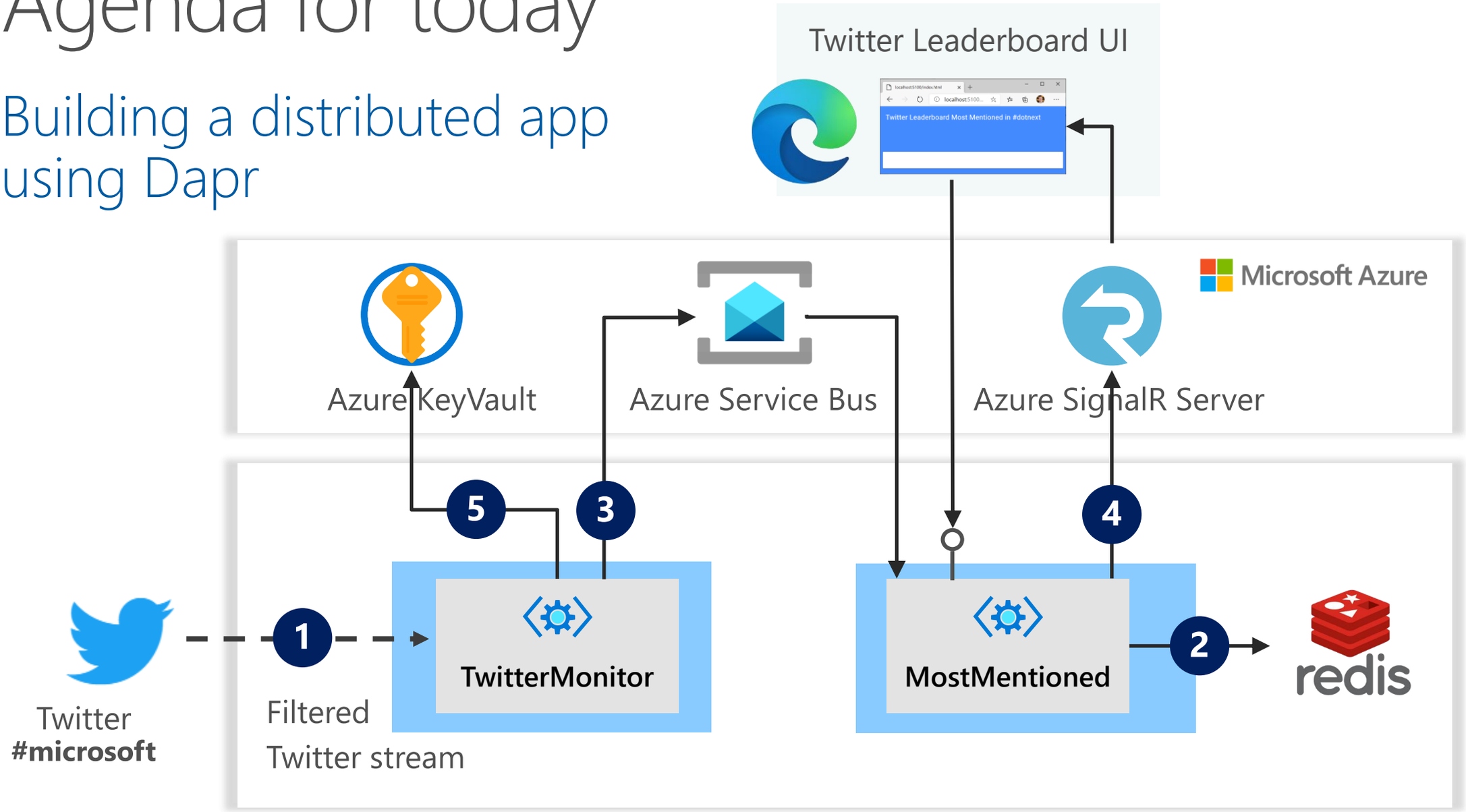
- Service to service invocation
- Actors

Components

- Publish and subscribe
- Resource bindings and triggers
- State Management
- Distributed tracing
- Secrets

Agenda for today

Building a distributed app using Dapr



Input and output bindings

Resource bindings and triggers

AWS

DynamoDB, Kinesis, S3, SNS, SQS

Azure

Blob Storage, CosmosDB, Event Grid, Event Hubs, Service Bus Queues, SignalR, Storage Queues

Google Cloud

Bucket, Pub/Sub

Other bindings

Alicloud OSS, HTTP, Kafka, Kubernetes, MQTT, RabbitMQ, Redis, Twillio SMS and SendGrid, Twitter

```yaml
apiVersion:
dapr.io/v1alpha1
kind: Component
metadata:
  name: tweets
spec:
  type: bindings.twitter
  metadata:
  - name: consumerKey
    value: "…"
  - name: consumerSecret
    value: "…"
  - name: accessToken
    value: "…"
  - name: accessSecret
    value: "…"
  - name: query
    value: "#microsoft"
```

Input binding

Resource bindings and triggers

Map to handler method

```
app.UseEndpoints(endpoints =>
{
  endpoints.MapPost("/tweets", HandleTwitterMessage);
});

async Task HandleTwitterMessage(HttpContext context)
{ … }
```

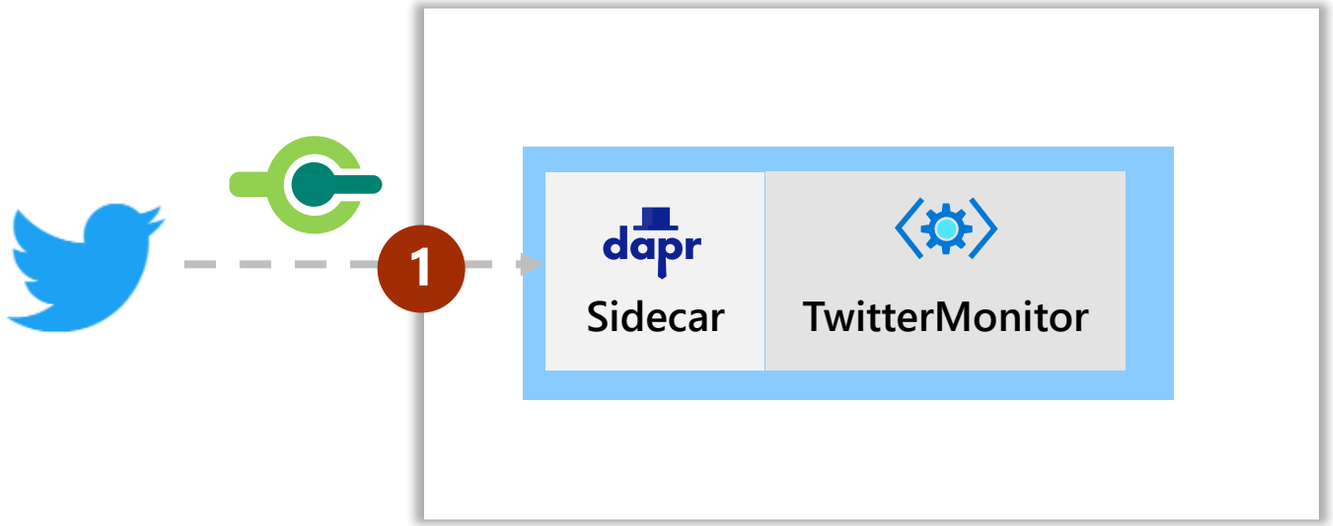
Map to controller method

Binding to model

```
[HttpPost("tweets")]
public async Task<ActionResult> Handle(Tweet tweet,
  [FromServices] DaprClient daprClient)
{
```

Demo

Input bindings

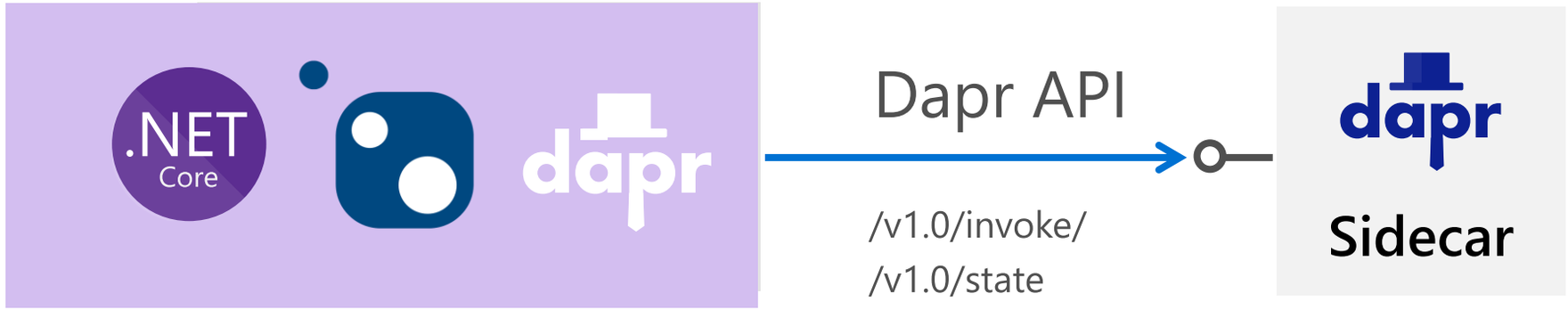


Dapr client

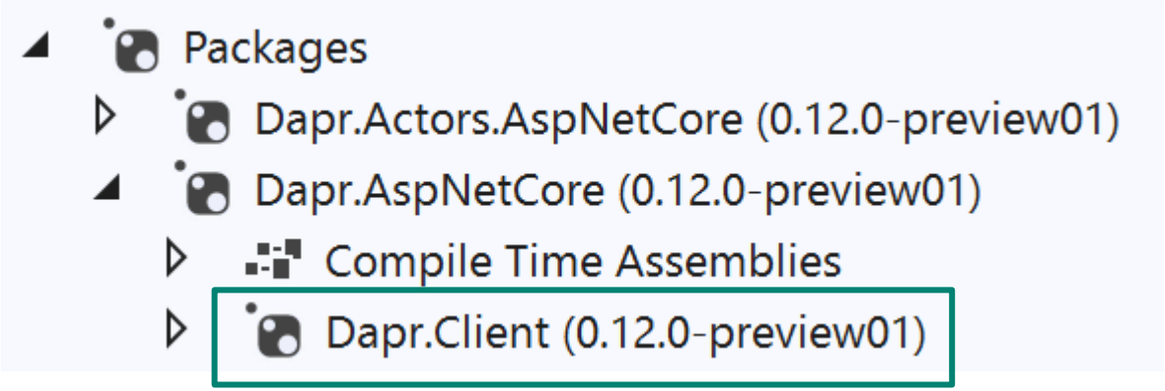
Strong typed proxy to sidecar

Invoke method or binding, use state, retrieve secrets and publish events

Uses gRPC endpoint



/v1.0/invoke/
/v1.0/state
/v1.0/publish
/v1.0/subscribe
/v1.0/bindings
/v1.0/actors
/v1.0/secrets



Dapr client

```
await daprClient.
```

- DeleteStateAsync
- GetSecretAsync
- GetStateAndETagAsync<>
- GetStateAsync<>
- GetStateEntryAsync<>
- InvokeBindingAsync<>
- InvokeMethodAsync
- InvokeMethodAsync<>
- PublishEventAsync
- PublishEventAsync<>
- SaveStateAsync<>
- TryDeleteStateAsync
- TrySaveStateAsync<>

Register in startup

```
services.AddDaprClient(config => {
  config.UseGrpcChannelOptions(new GrpcChannelOptions() {
    DisposeHttpClient = true });
  config.UseJsonSerializationOptions(new JsonSerializerOptions() {
    PropertyNamingPolicy = JsonNamingPolicy.CamelCase,
    PropertyNameCaseInsensitive = true
  });
});
```

Resolve with dependency injection

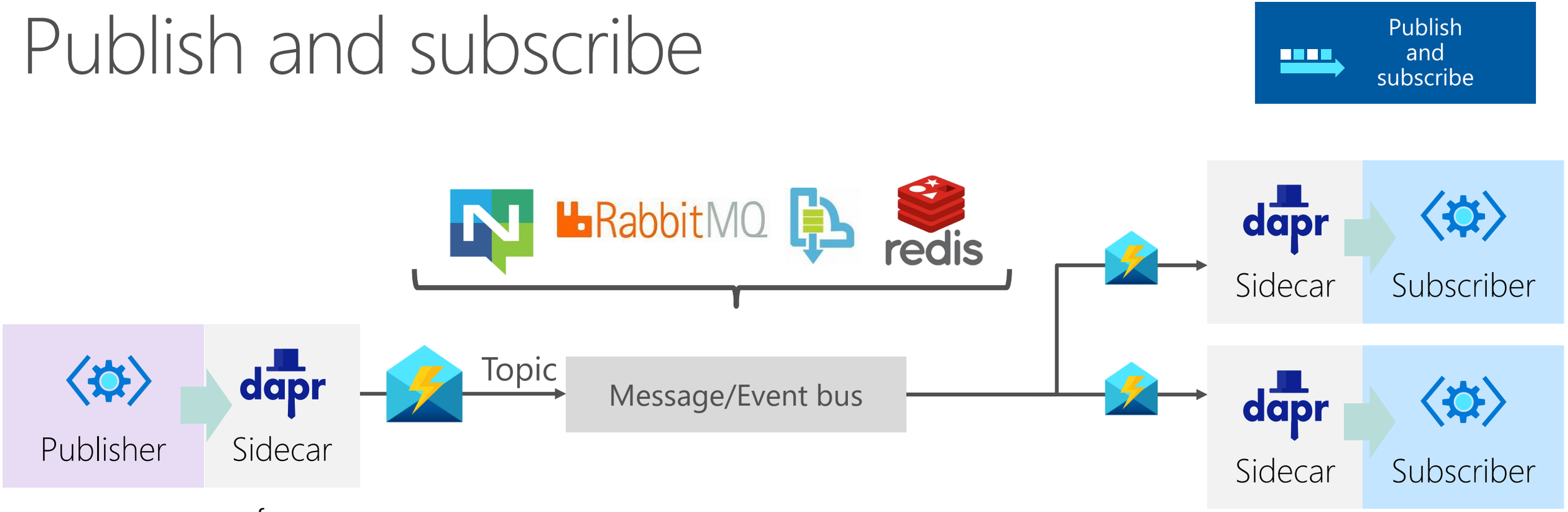
Constructor injection

```
public TweetController(DaprClient client)
```

Per method injection

```
public async Task<ActionResult<Tweet>> Handle([FromServices] DaprClient daprClient)
```

Publish and subscribe



1 or multiple subscribers


```
{
  "id": "a50e54f6-24c5-4c2e-893e-84044fc27e5b",
  "source": "monitor",
  "type": "com.dapr.event.sent",
  "specversion": "1.0",
  "datacontenttype": "application/json",
  "data": {
    "TweetId": 1235673137997656064,
    "UserName": "alexthissen"
  },
  "subject": "00-e6e9d28172535b66e9a9840a8b8e01d9-cb3dc01baa833db5-01"
}
```


CloudEvents 1.0 specification for encapsulation of message

Publish and subscribe

Publishing a message

Standalone mode with **Dapr CLI**

Using Dapr client in code

```
dapr publish --topic tweet --pubsub messagebus
        --data "{ 'payload':'yourdata' }"
```

```
await daprClient.PublishEventAsync<Tweet>("messagebus", "tweet", new Tweet { … });
```

Preparing for subscriptions

Dapr runtime expects a special endpoint for subscriptions

```
app.UseCloudEvents();
app.UseEndpoints(endpoints =>
{
  endpoints.MapSubscribeHandler();
  endpoints.MapControllers();
});
```

Register middleware that unwraps message payload based on CloudEvents 1.0 encapsulation

Collects all route/topic subscriptions and creates **/dapr/subscribe** endpoint

Subscribing to events

Splitting R **route** from T **topic**

Explicit endpoint mapping to handler

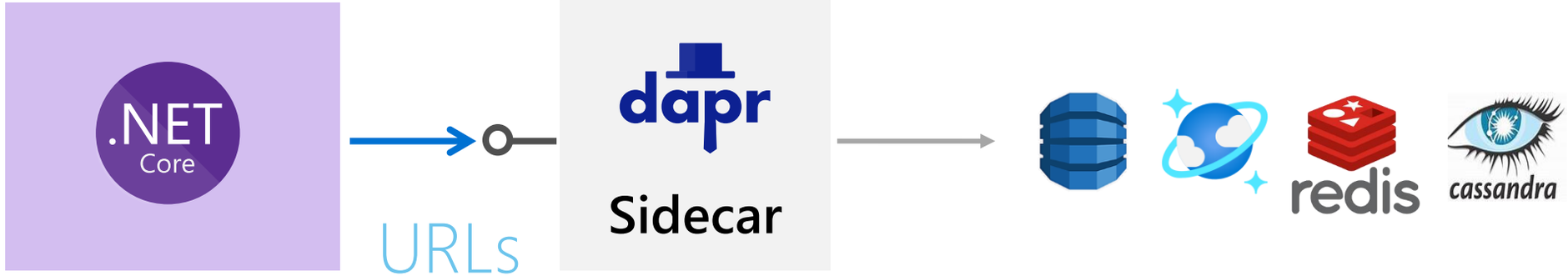
```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
  app.UseEndpoints(endpoints => {
    endpoints.MapPost("tweet", ReceiveTweet).WithTopic("tweets");
    endpoints.MapSubscribeHandler();
    endpoints.MapControllers();
}
```

Controller actions

```
[Topic("messagebus", "tweets")]
[HttpPost("tweet")]
public async Task<ActionResult<Tweet>> Handle(Tweet tweet)
{
```

State storage

State Management



URLs

POST http://localhost:{daprport}/v1.0/state/{storename}
GET/DELETE http://localhost:{daprport}/v1.0/state/{storename}/{key}

Stores

Multiple stores are allowed. Separate YAML file for each

```yaml
apiVersion: dapr.io/v1alpha1
kind: Component
metadata:
  name: statestore
spec:
  type: state.redis
  metadata:
  - name: redisHost
    value: redis:6379
  - name: redisPassword
    value: ""
  - name: actorStateStore
    value: "true"
```

Store name

Used in code to refer to store

State options

Consistency
Eventually or strong
Optimistic Concurrency Control (OCC) using Etags
Resiliency
Retry interval (in milliseconds),
Retry pattern (linear or exponential),
Retry threshold (maximum number of retries)

.NET Storage with Dapr

State Management

FromState and StateEntry

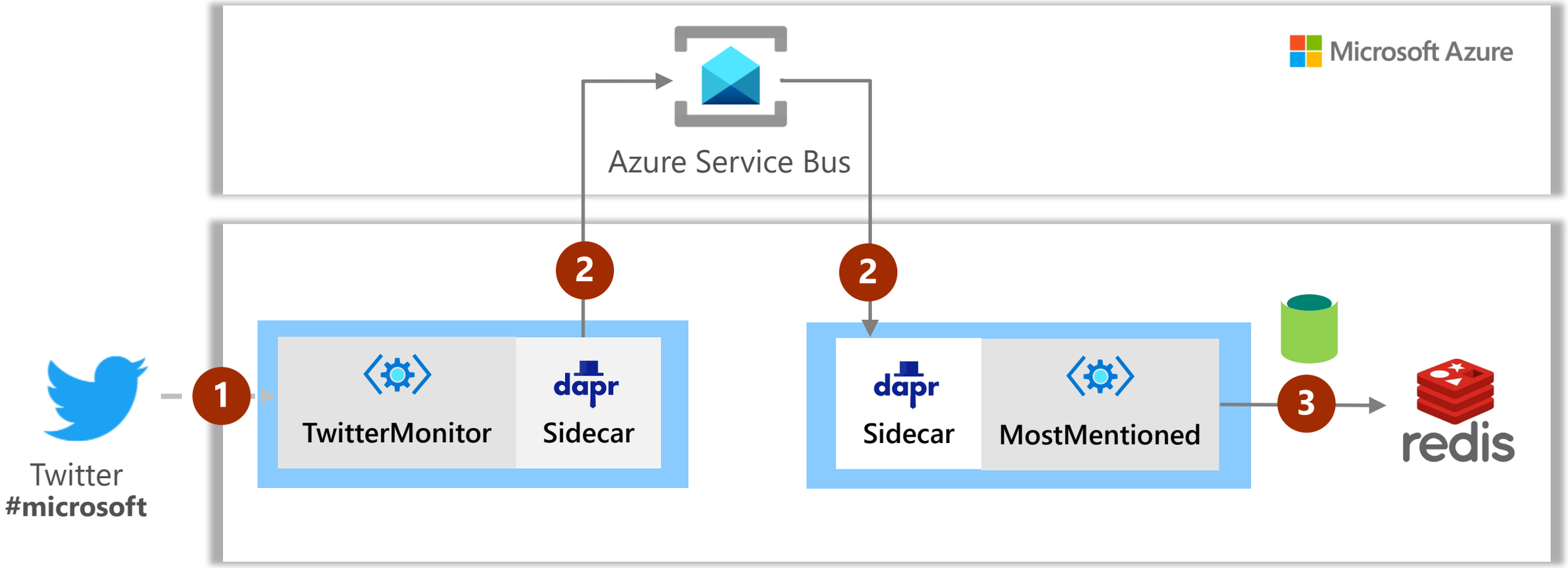
```
[HttpGet("{tweet}")]
public ActionResult<Tweet> Get([FromState(StoreName)]StateEntry<Tweet> tweet)
{ … }
```

DaprClient

```
[HttpPost("tweet")]
public async Task<ActionResult<Tweet>> Handle([FromServices] DaprClient daprClient)
{
```

Demo

State storage and pub/sub



Concepts for actors

Concepts for actors

Actors have an identity (Actor ID)

Turn-based access:

- Process one request at a time
- Respected across different methods, timers and callbacks

Actors will be garbage collected by inactivitity

Reminder firing will keep actor active

Idle timeout and scan interval configurable

Some details for actors

Only one state store can store actor data

Redis

MongoDB

SQL Server

```yaml
apiVersion: dapr.io/v1alpha1
kind: Component
metadata:
  name: statestore
spec:
  type: state.redis
  metadata:
  - name: redisHost
    value: localhost:6379
  - name: redisPassword
    value: ""
  - name: actorStateStore
    value: "true"
```

Actor placement service

Calculates partitioning across instances per actor type

Randomly placed across pods (expect network traffic)

Used only for Dapr actor placement. Not needed when no actors are used

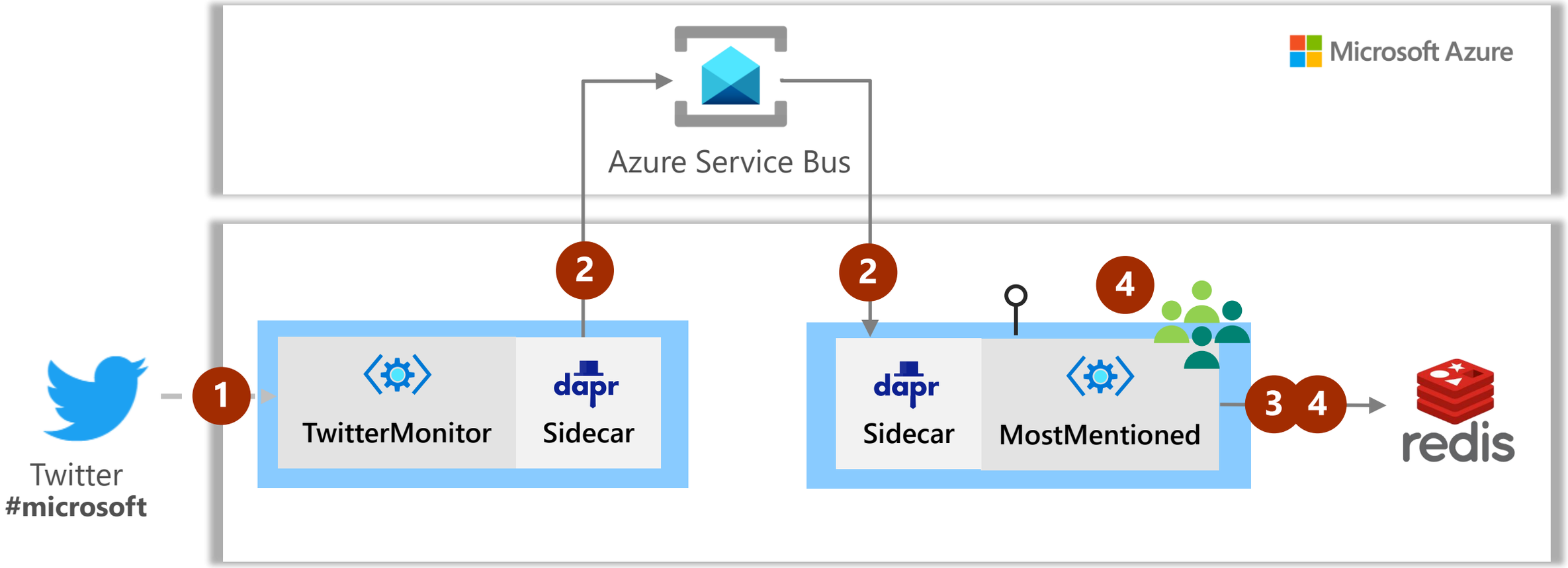
Calling Dapr actor in sidecar

POST/GET/PUT/DELETE

http://localhost:3500/v1.0/actors/<actorType>/<actorId>/method/<method>

Demo

Actors


Microsoft Azure
Azure Service Bus
2
2
4
1
TwitterMonitor
dapr
Sidecar
dapr
Sidecar
MostMentioned
3
4
redis
Twitter
#microsoft

Two way output binding

```yaml
apiVersion: dapr.io/v1alpha1
kind: Component
metadata:
  name: signalr
  namespace: default
spec:
  type: bindings.azure.signalr
  metadata:
  - name: connectionString
    value: Endpoint=…;AccessKey=…;Version=1.0;
  - name: hub
    value: leaderboardhub
```

Invoking using Dapr client

```
await daprClient.InvokeBindingAsync<Data>("signalr", "create", data);
```

Output bindings

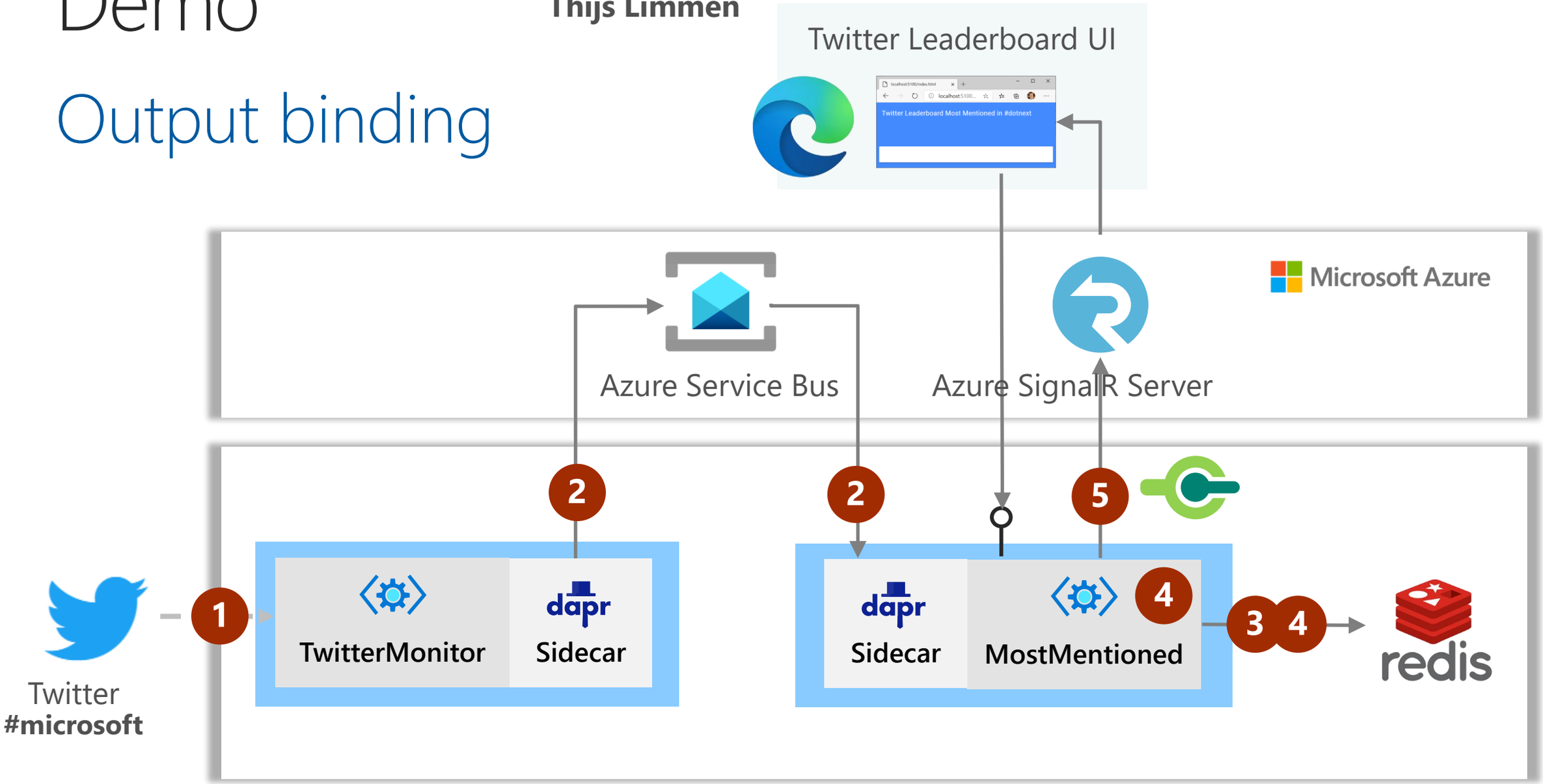
aws.sqs
aws.kinesis
azure.eventhubs
kafka
mqtt
rabbitmq
azure.servicebusqueues
azure.storagequeues
gcp.pubsub
azure.eventgrid

aws.dynamodb
azure.cosmosdb
gcp.bucket
http
redis
aws.s3
aws.sns
azure.blobstorage
azure.signalr
twilio.sms
twilio.sendgrid

Demo

Output binding

VueJS frontend by
Thijs Limmen



Setting up tracing

```yaml
apiVersion: dapr.io/v1alpha1
kind: Component
metadata:
  name: zipkin
  namespace: default
spec:
  type: exporters.zipkin
  metadata:
  - name: enabled
    value: "true"
  - name: exporterAddress
    value: "http://localhost:9411/api/v2/spans"
```

```yaml
apiVersion: dapr.io/v1alpha1
kind: Configuration
metadata:
  name: myconfig
  namespace: default
spec:
  tracing:
    samplingRate: "1"
```

OpenCensus probalistic sampling

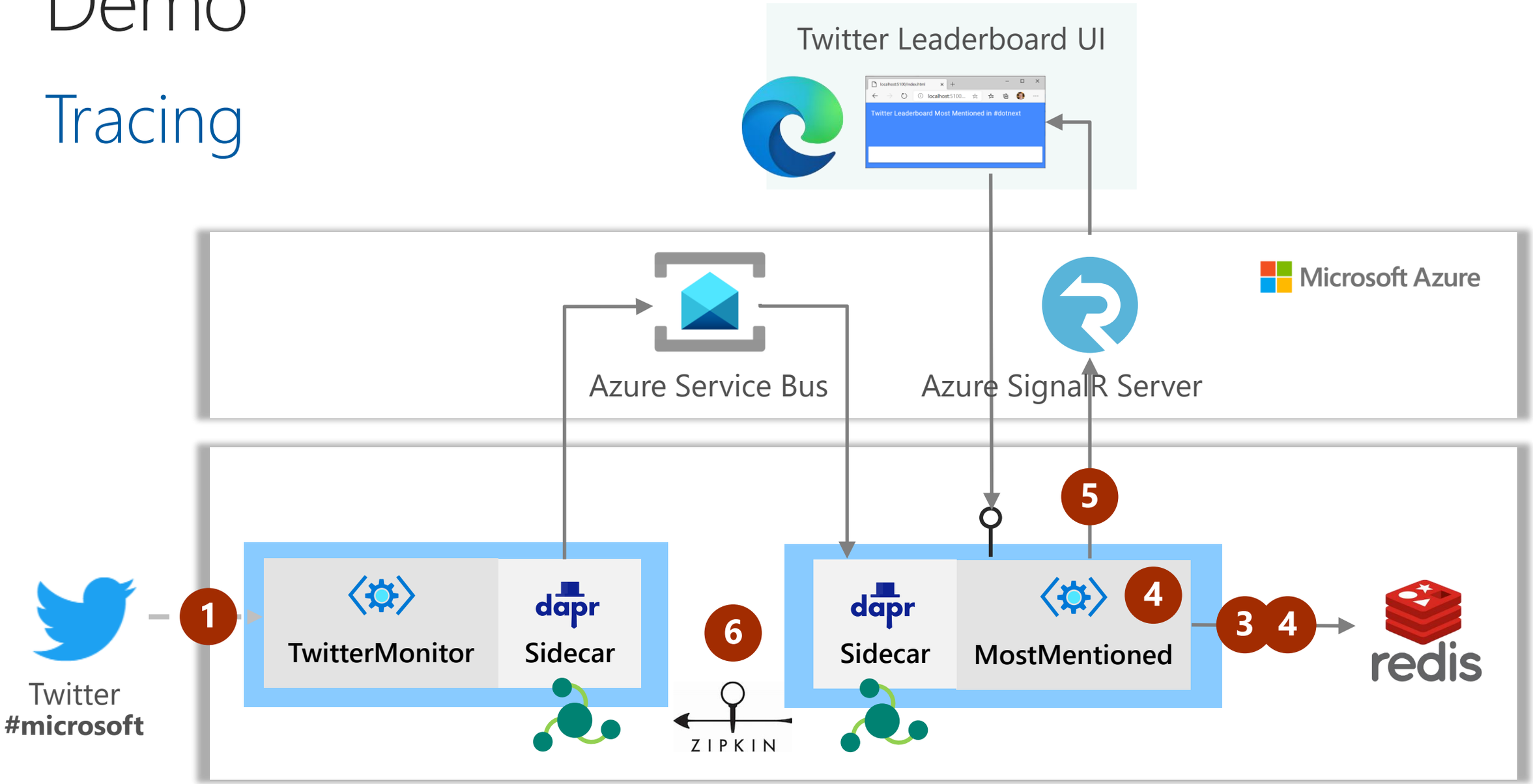
Defines percentage of samples taken

Use Dapr configuration file during bootstrap

```
dapr run --app-id dotnext --app-port 5000 --config .\components\tracing.yaml dotnet run
```

Demo

Tracing



Summarizing and next steps

Accelerate developing distributed apps

Sidecar takes away heavy lifting

Building blocks and capabilities decrease learning time

Portability by cloud provider agnostic runtime

Run on any cloud platform

Use components from arbitrary platform

Switching requires no code changes*

Dapr is evolving

Extending Dapr by community contributions and feedback to team

Help by contributing

Resources

Dapr on GitHub

https://github.com/dapr
https://github.com/dapr/docs
https://github.com/dapr/samples
https://github.com/dapr/dotnet-sdk

Demo source code

https://github.com/alexthissen/DaprNETCore

Community

@daprdev
Bi-weekly community calls
https://gitter.im/Dapr/

Questions and Answers

Maybe later?

mdevries@xpirit.com
athissen@xpirit.com

Xpirit

Microsoft® Most Valuable Professional