KubeCon

CloudNativeCon

North America 2024

Scale Job Triggering with a Distributed Scheduler

Artur Souza & Cassie Coyle
Diagrid

Table of Contents

Speakers

Diagrid

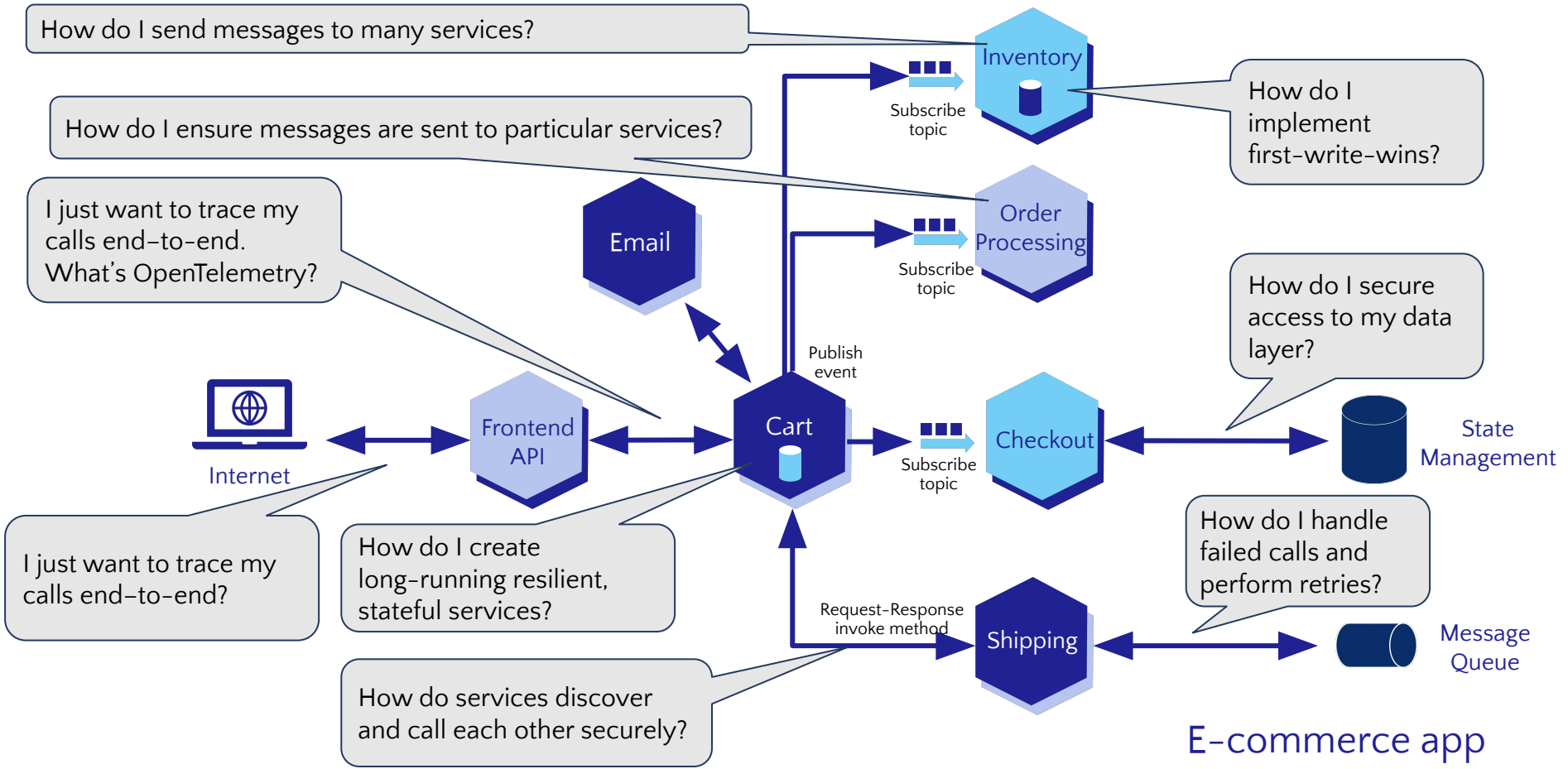
dapr

Software Engineer
@cicoyle
https://www.linkedin.com/in/cassie-coyle/

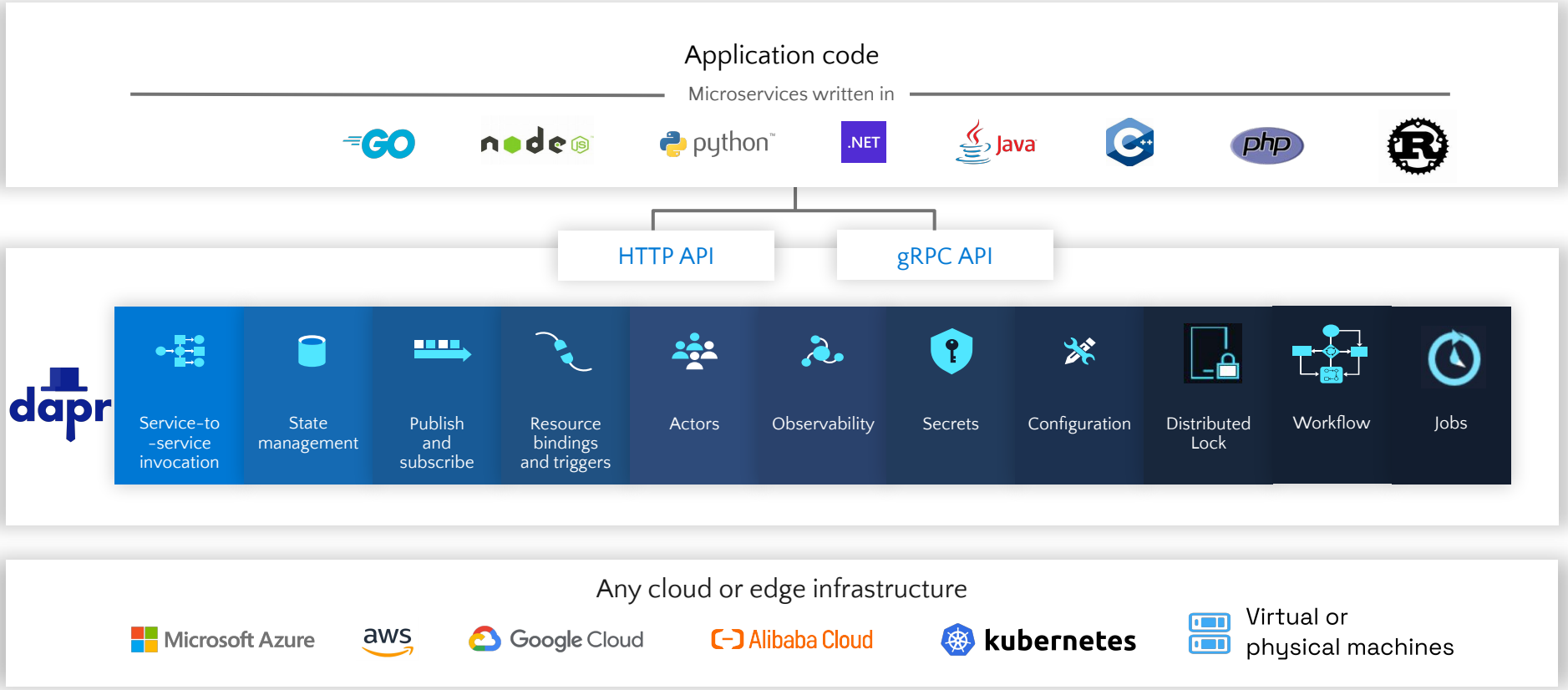
Head of Engineering
@artursouza
https://www.linkedin.com/in/barbalho/

Introduction to Dapr

Developer Challenges



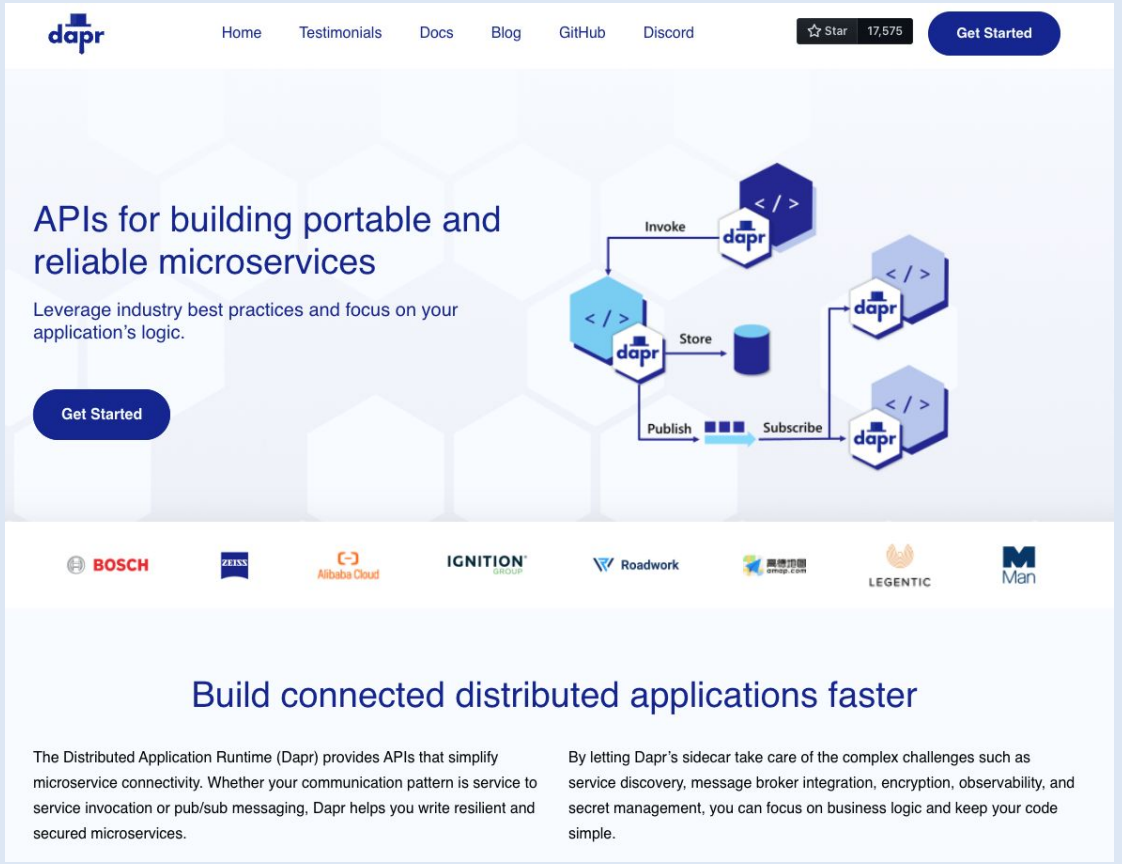
Dapr APIs


Application code
Microservices written in
HTTP API
gRPC API
dapr
Service-to-service invocation
State management
Publish and subscribe
Resource bindings and triggers
Actors
Observability
Secrets
Configuration
Distributed Lock
Workflow
Jobs
Any cloud or edge infrastructure
Microsoft Azure
aws
Google Cloud
Alibaba Cloud
kubernetes
Virtual or physical machines

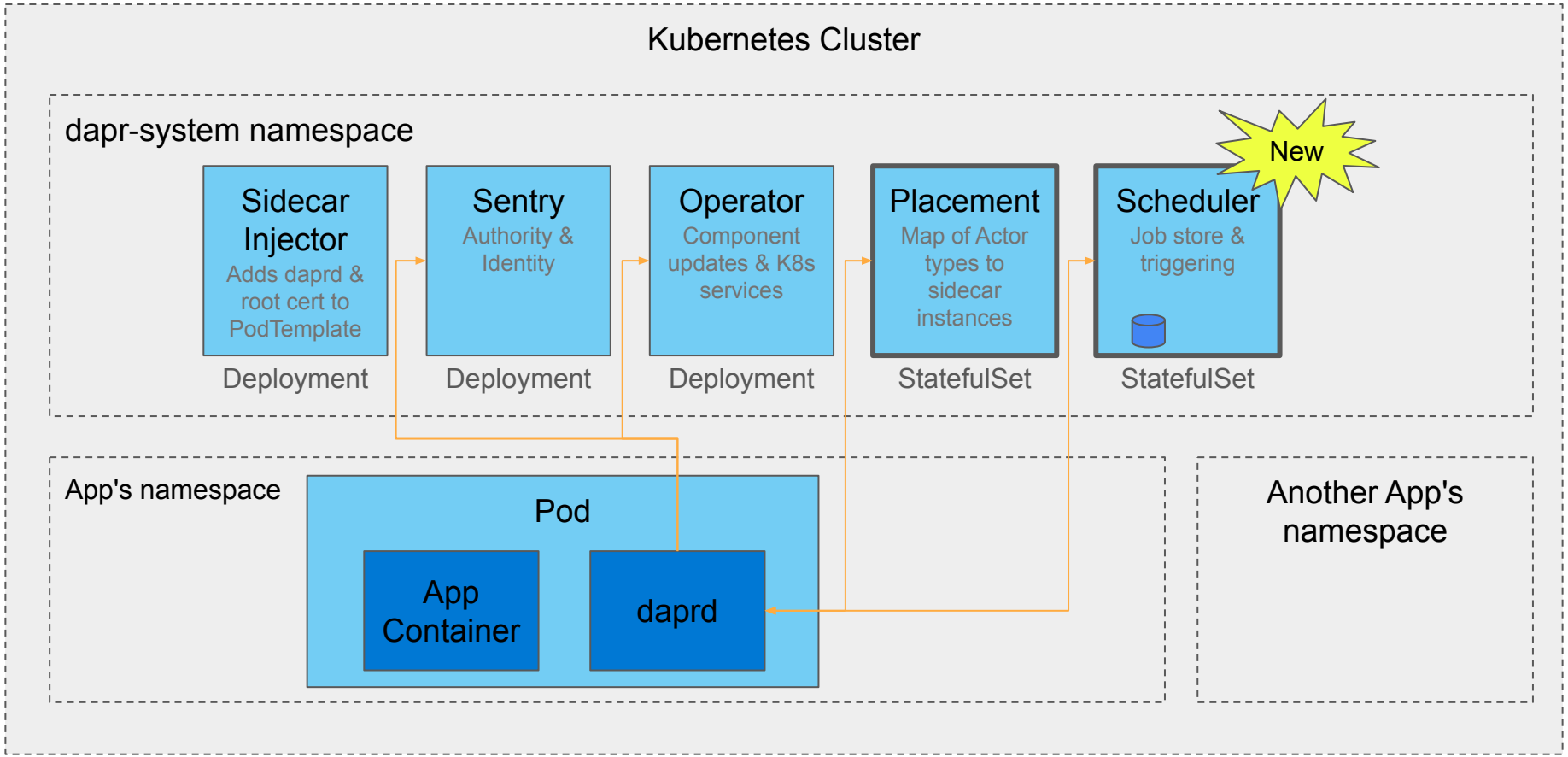
Distributed Application Runtime

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

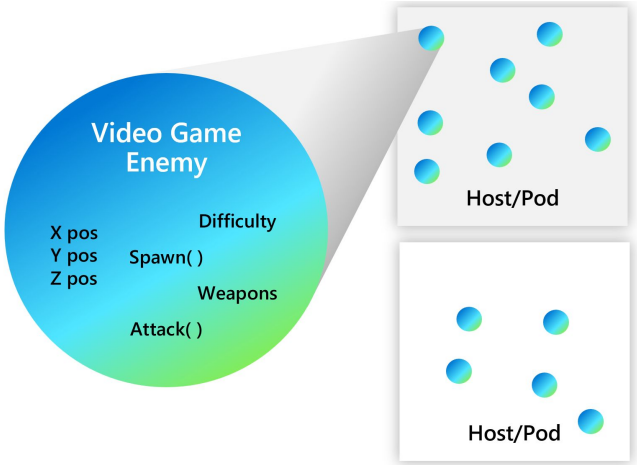
dapr.io

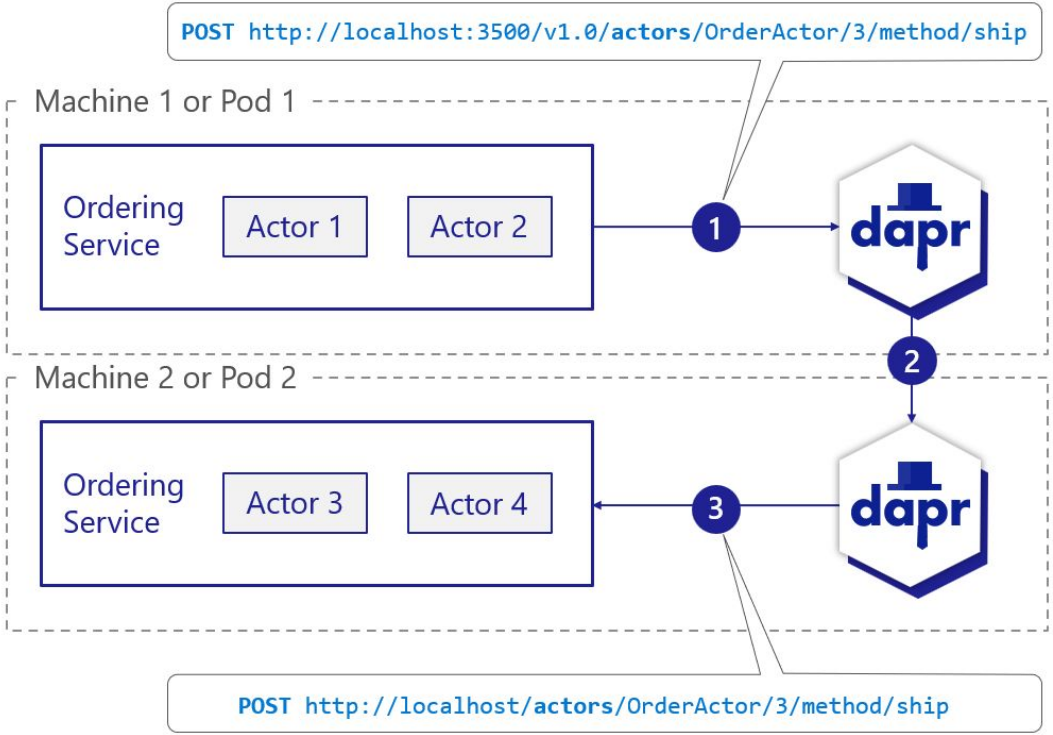


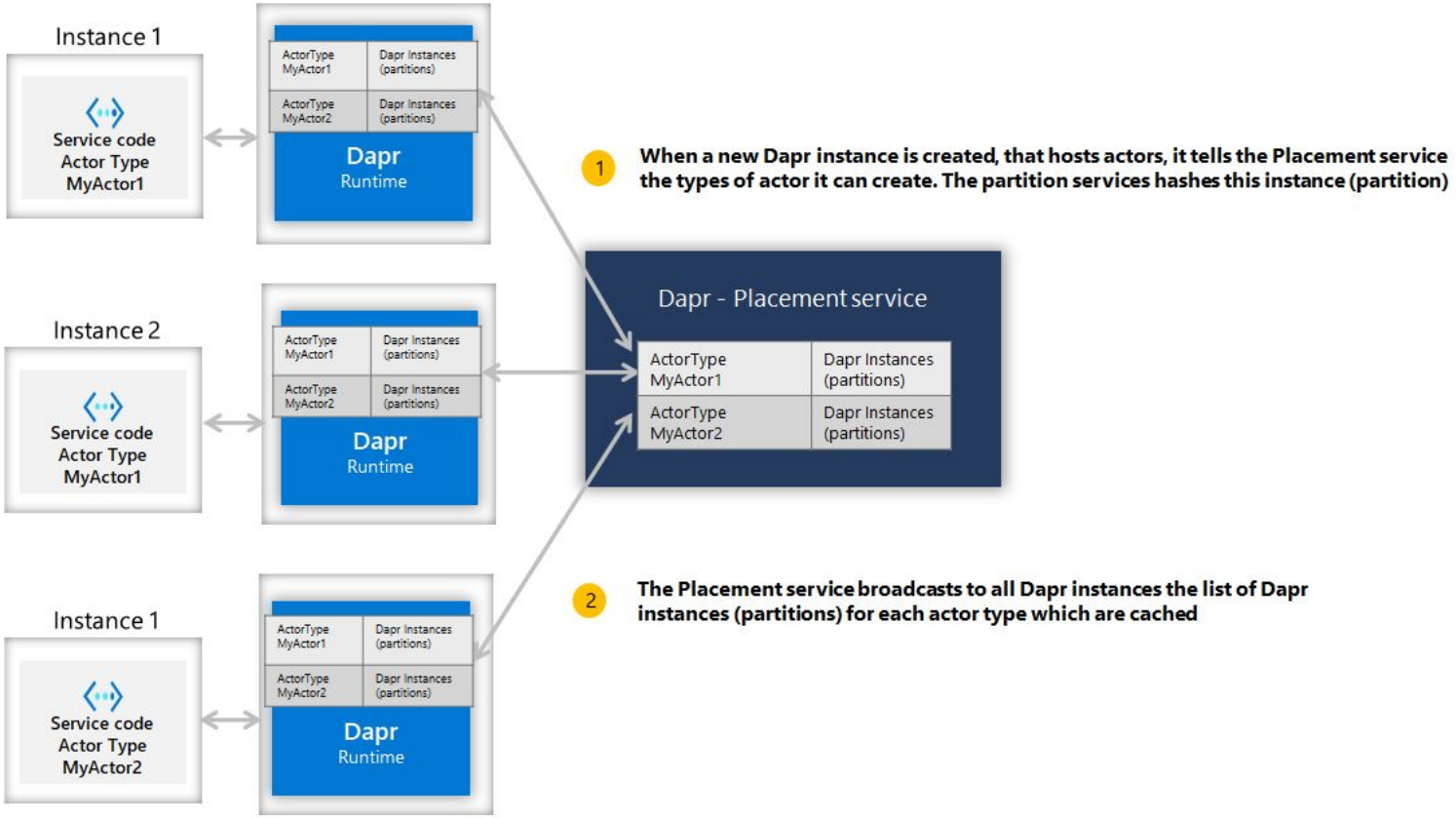
Dapr’s Architecture


Kubernetes Cluster
dapr-system namespace
New
Sidecar Injector
Adds daprd & root cert to PodTemplate
Deployment
Sentry
Authority & Identity
Deployment
Operator
Component updates & K8s services
Deployment
Placement
Map of Actor types to sidecar instances
StatefulSet
Scheduler
Job store & triggering
StatefulSet
App's namespace
Pod
App Container
daprd
Another App's namespace

Actors

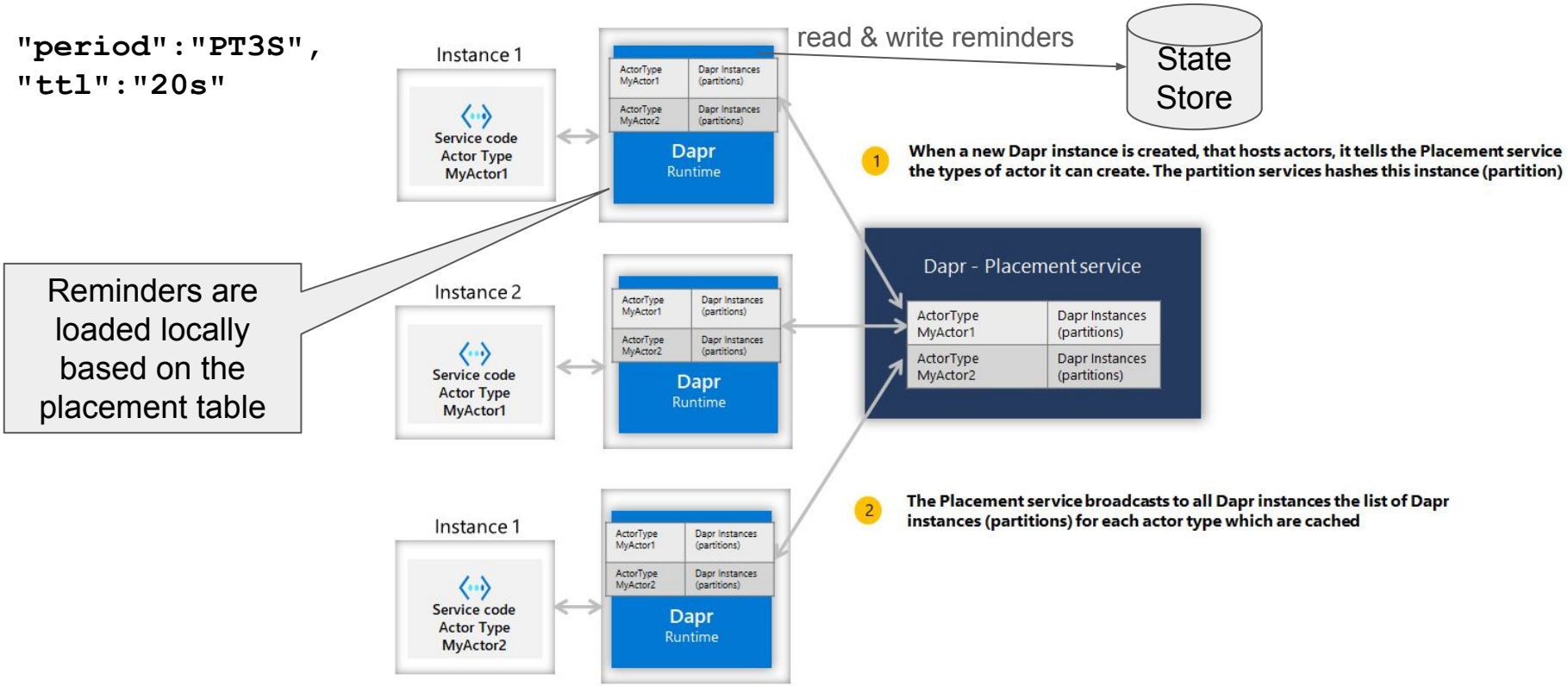

Video Game Enemy
X pos
Y pos
Z pos
Difficulty
Spawn( )
Weapons
Attack( )
Host/Pod
Host/Pod


POST http://localhost:3500/v1.0/actors/OrderActor/3/method/ship
Machine 1 or Pod 1
Ordering Service
Actor 1
Actor 2
1
dapr
2
Machine 2 or Pod 2
Ordering Service
Actor 3
Actor 4
3
dapr
POST http://localhost/actors/OrderActor/3/method/ship


Instance 1
Service code
Actor Type
MyActor1
ActorType MyActor1
Dapr Instances (partitions)
ActorType MyActor2
Dapr Instances (partitions)
Dapr
Runtime
Instance 2
Service code
Actor Type
MyActor1
ActorType MyActor1
Dapr Instances (partitions)
ActorType MyActor2
Dapr Instances (partitions)
Dapr
Runtime
Instance 1
Service code
Actor Type
MyActor2
ActorType MyActor1
Dapr Instances (partitions)
ActorType MyActor2
Dapr Instances (partitions)
Dapr
Runtime
Dapr - Placement service
ActorType MyActor1
Dapr Instances (partitions)
ActorType MyActor2
Dapr Instances (partitions)
1
When a new Dapr instance is created, that hosts actors, it tells the Placement service the types of actor it can create. The partition services hashes this instance (partition)
2
The Placement service broadcasts to all Dapr instances the list of Dapr instances (partitions) for each actor type which are cached

Actor Reminders

```
POST/PUT http://localhost:3500/v1.0/actors/<actorType>/<actorId>/reminders/<name>
{
  "period":"PT3S",
  "ttl":"20s"
}
```



Actors||Customer

```
[
  {
    "actorType": "Customer",
    "actorId": "100",
    "name": "paymentReminder",
  },
  {
    "actorType": "Customer",
    "actorId": "200",
    "name": "paymentReminder",
  },
]
```

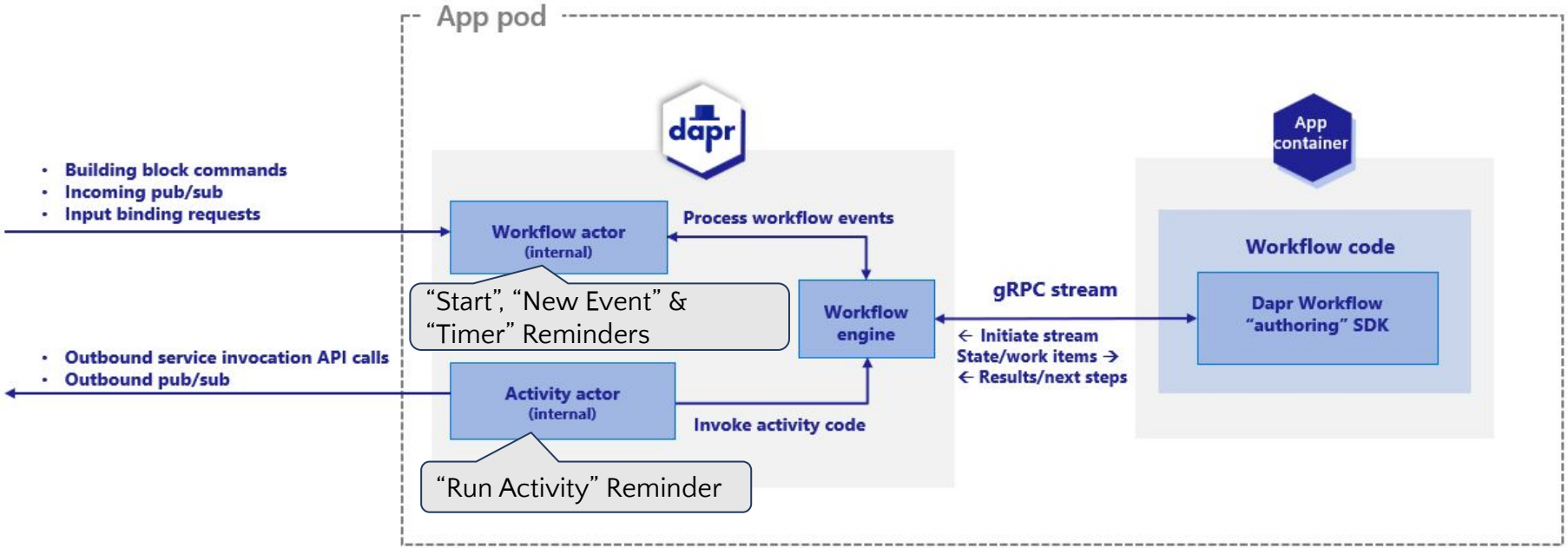
Key	Value
actors||<actor type>||metadata	{ "id": <actor metadata identifier>, "actorRemindersMetadata": { "partitionCount": <number of partitions for reminders> } }
actors||<actor type>||<actor metadata identifier>||reminders||1	[<reminder 1-1>, <reminder 1-2>, ... , <reminder 1-n>]
actors||<actor type>||<actor metadata identifier>||reminders||2	[<reminder 1-1>, <reminder 1-2>, ... , <reminder 1-m>]

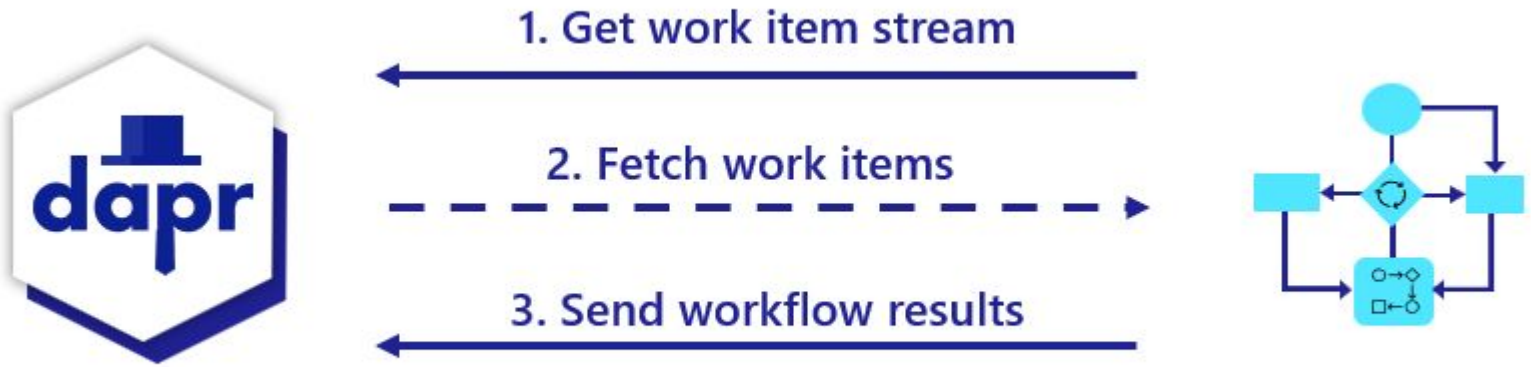
Key	Value
actors||**Customer**||metadata	{ "id": **"64d9c7be-8f46-4e1b-9d8a-a95a8aabb43e"**, "actorRemindersMetadata": { "partitionCount": **2** } }
actors||**Customer**||**64d9c7be-8f46-4e1b-9d8a-a95a8aabb43e**||reminders||**1**	[{...}, {...}, {...}]
actors||**Customer**||**64d9c7be-8f46-4e1b-9d8a-a95a8aabb43e**||reminders||**2**	[{...}, {...}, {...}, {...}]

Limitations with Actor Reminders

- Low throughput to register or delete reminders: ~45 tps
- Cannot scale throughput horizontally or vertically
- Limited number of reminders that can be registered: ~1,000 practical limit
- Rebalance required when application pods go up or down

Dapr Workflow


App pod
dapr
App container
• Building block commands
• Incoming pub/sub
• Input binding requests
Workflow actor (internal)
Process workflow events
"Start", "New Event" & "Timer" Reminders
Workflow engine
gRPC stream
← Initiate stream
State/work items →
← Results/next steps
Workflow code
Dapr Workflow "authoring" SDK
• Outbound service invocation API calls
• Outbound pub/sub
Activity actor (internal)
Invoke activity code
"Run Activity" Reminder


dapr
1. Get work item stream
2. Fetch work items
3. Send workflow results

Limitations with Workflows

Practical limit of concurrent workflow activities: ~100

Practical limit of concurrent workflows: 2

Dapr Scheduler Service

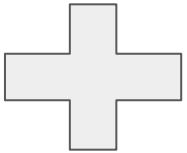
Dapr Scheduler Service

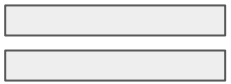
- New control plane service
  - Deployed by default with *dapr init* CLI
  - Run in single instance or in HA mode
- Capabilities
  - Stores jobs to be triggered at some point in the future
  - Guarantees that a job is triggered by one Scheduler
- Implementation
  - Embedded etcd database
  - Internal cron scheduling library

Design Decisions

- Job orchestrator, not executor
- At least once job execution
- Bias towards durability and horizontal scaling over clock-time precision
- Generic for multi-purpose job usage

Design Decisions

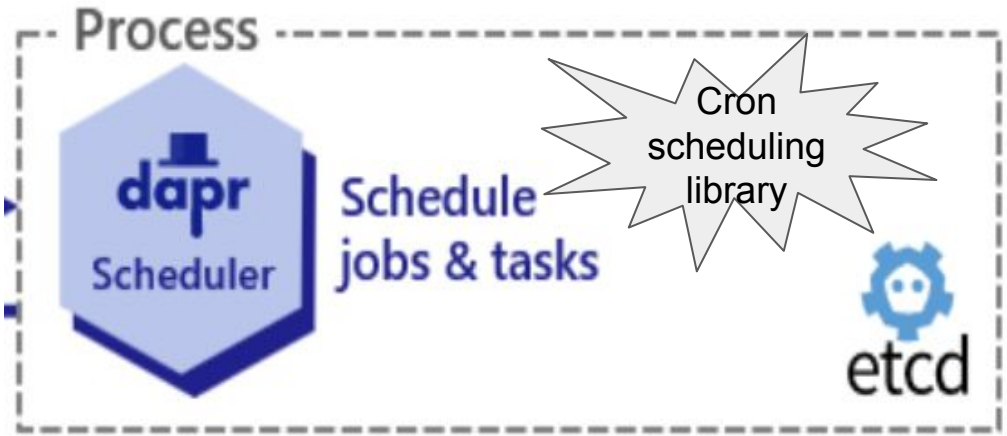




How Does it Work?

Architecture

Architecture

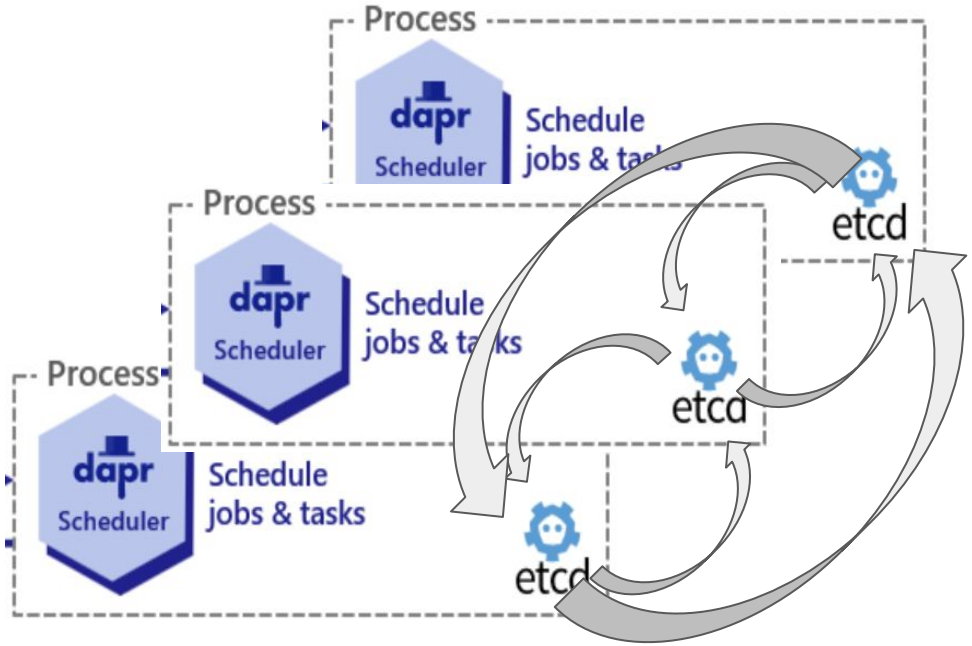


Scheduler Server


```
err = concurrency.NewRunnerManager(
    …
    func(ctx context.Context) error {
        …
        server, serr := server.New(server.Options{
            Port:           opts.Port,
            ListenAddress: opts.ListenAddress,
            Mode:           modes.DaprMode(opts.Mode),
            Security:       secHandler,
            Healthz:        healthz,
            DataDir:             opts.EtcdDataDir,
            ReplicaCount:        opts.ReplicaCount,
            ReplicaID:           opts.ReplicaID,
            KubeConfig:          opts.KubeConfig,
            EtcdID:             opts.ID,
            EtcdInitialPeers:   opts.EtcdInitialPeers,
            EtcdClientPorts:    opts.EtcdClientPorts,
            … (etcd config)
        })
        …
        return server.Run(ctx)
    }).Run(ctx)
```

Embedded etcd

- Distributed KV store
- State management of jobs
- Data consistency



Start Embedded etcd


```
import "go.etcd.io/etcd/server/v3/embed"
…
etcd, err := embed.StartEtcd(c.config)
if err != nil {
    return err
}
defer etcd.Close()

select {
case <-etcd.Server.ReadyNotify():
    log.Info("Etcd server is ready!")
case <-ctx.Done():
    return ctx.Err()
}
```

etcd Data: Leadership

etcd data: Replicated across all instances. The same data for all instances.	
`dapr/leadership/0`	3
`dapr/leadership/1`	3
`dapr/leadership/2`	3

etcd Data: Jobs API

etcd data: Replicated across all instances. The same data for all instances.

`dapr/jobs/app||namespace||appid||jobid`	`val`

etcd Data: Actor Reminders

etcd data: Replicated across all instances. The same data for all instances.

`dapr/jobs/actorreminder||default||myactortype||myactorid||remindermethod`	`val`

etcd Data: Workflow Actor Reminders

etcd data: Replicated across all instances. The same data for all instances.

dapr/jobs/actorreminder||default||dapr.internal.default.wf-app.workflow||wf-actorid||start-eqqANOKQ	val
dapr/jobs/actorreminder||default||dapr.internal.default.wf-app.activity||wf-actorid::0::1||run-activity	val
dapr/jobs/actorreminder||default||dapr.internal.default.wf-app.workflow||wf-actorid||new-event-auiNE1fw	val

etcd Data

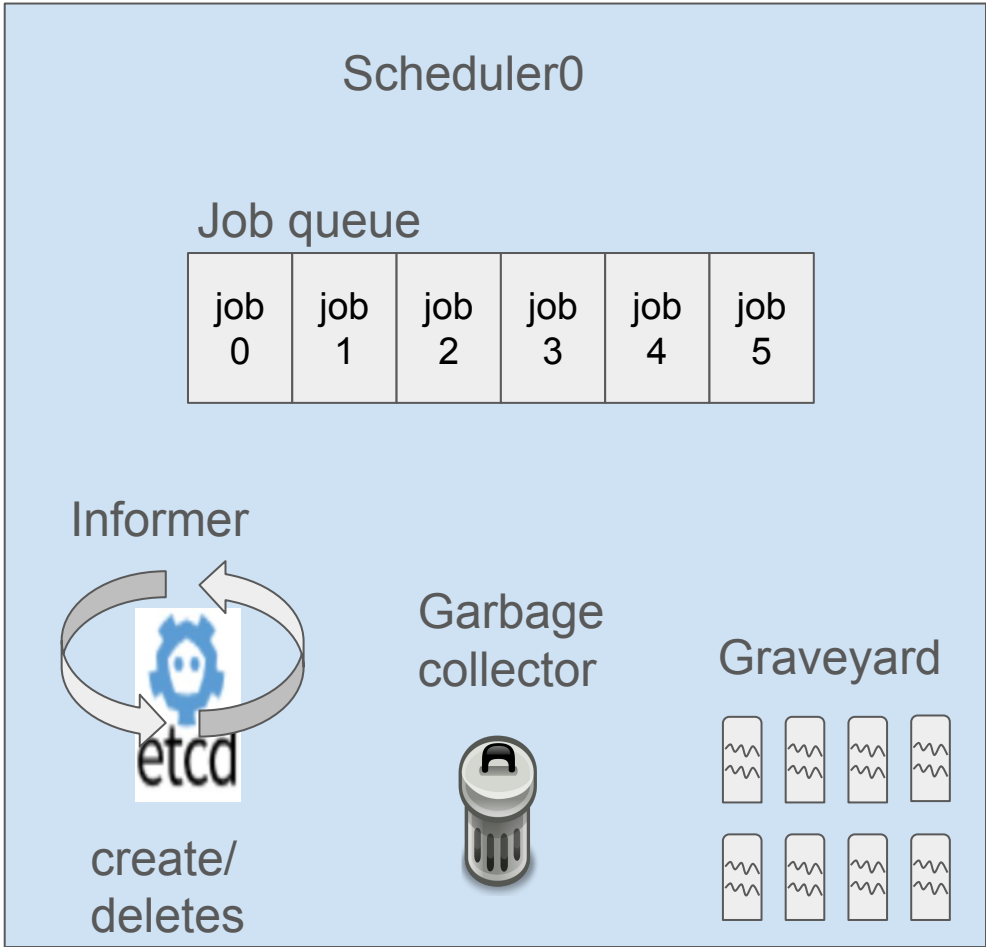
etcd data: Replicated across all instances. The same data for all instances.	
dapr/leadership/0	3
dapr/leadership/1	3
dapr/leadership/2	3
dapr/jobs/app||namespace||appid||jobid	val
dapr/jobs/actorreminder||default||myactortype||myactorid||remindermethod	val
dapr/jobs/actorreminder||default||dapr.internal.default.wf-app.workflow||wf-actorid||start-eqqANOKQ	val
dapr/jobs/actorreminder||default||dapr.internal.default.wf-app.activity||wf-actorid::0::1||run-activity	val
dapr/jobs/actorreminder||default||dapr.internal.default.wf-app.workflow||wf-actorid||new-event-auiNE1fw	val

Internal Cron Scheduling Library

- Enable scalable, distributed job management
- Dynamic job partition leadership coordination
- Enables load distribution

Cron pkgs

- Queue
  - Manages the scheduling and triggering of jobs
- Counter
  - Track state of triggered jobs over time
- Leadership
  - Job ownership
- Informer
  - Watches for changes in the job keyspace
- Graveyard
  - Track & discard keys
- Garbage Collector
  - Bulk delete keys

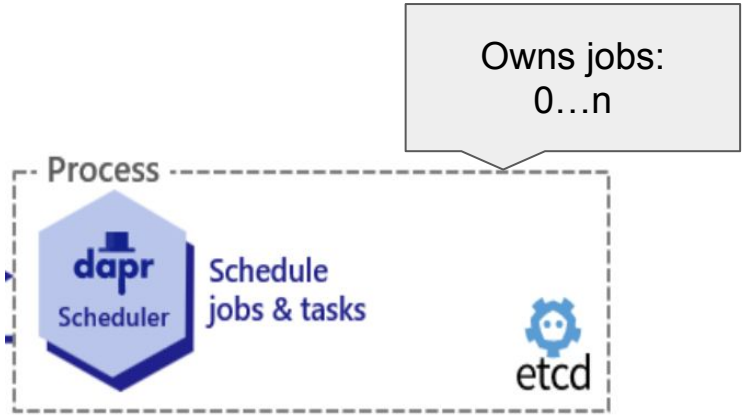


Internal Cron Scheduling Library

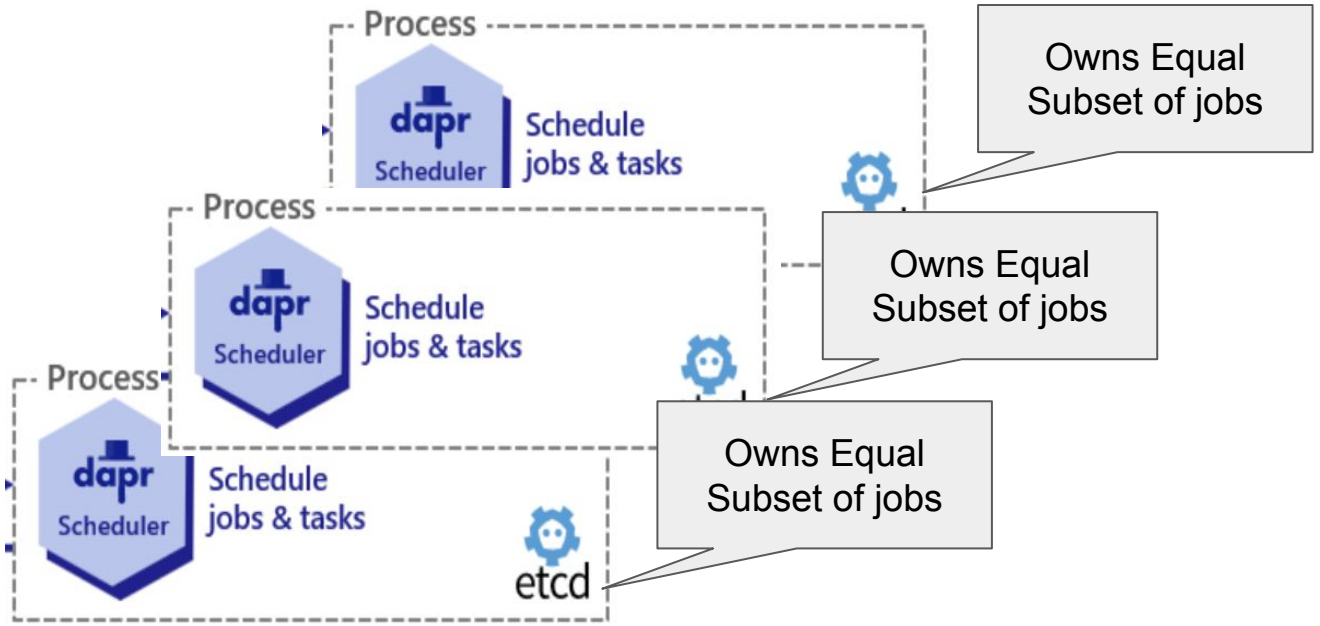

```
import "github.com/diagridio/go-etcd-cron/cron "
…
cron, err := cron.New(cron.Options{
        Client:         client,
        Namespace:      "dapr",
        PartitionID:    c.replicaID,
        PartitionTotal: c.replicaCount,
        TriggerFn:      c.triggerJob,
        ReplicaData:    replicaData,
})

go cron.Run(context.Background)
```

Cron Library: Ownership model



Cron Library: Ownership model




```
partitionID%m.totalPartitions
```

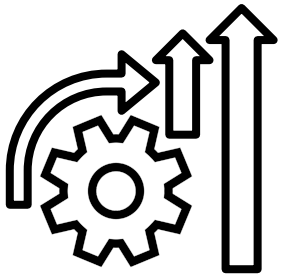
Scheduler Resiliency

- Data persistence & replication with etcd
- Dynamic job partition leadership & job distribution
- Jobs are always triggered using the persisted counter for catch-up
- Failure Policy & staging queue

Impact on Actor Reminders and Workflow

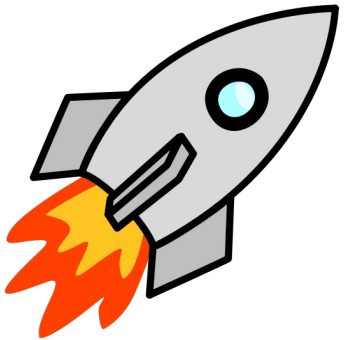
Actor Reminders Performance Using Scheduler

- Performance gains in HA mode (3 Scheduler instances)
  - Schedules 50,000 actor reminders with an average trigger QPS of 4,582
  - At least a 10x improvement with stable QPS
  - Direct API invocations achieves up to 35,000 QPS
- Drastic improvements over Dapr v1.13 while creating actor reminders
  - Dapr v1.13 QPS: 50
  - Dapr v1.14 QPS: 4,000 (an 80x increase) with Scheduler



Workflow Performance Numbers Using Scheduler

- Parallel Workflow Testing:
  - Max Concurrent Count (60-90):
    - Performance Improvement: 71%
    - Existing Reminder System: Drops by 44%
- High Scale Testing:
  - Max Concurrent Workflows: 350
  - Iterations: 1400
  - Performance Improvement: 50% higher than existing reminder system
- Scale to millions of reminders



Dapr Jobs API

Dapr Jobs API

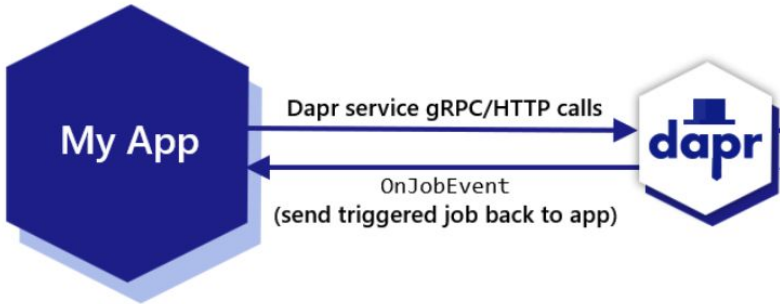
- Alpha API
- Schedule/Get/Delete

```
curl -X POST \
  http://localhost:3500/v1.0-alpha1/jobs/test
\
  -H "Content-Type: application/json" \
  -d '{
        "data": "cassie",
        "dueTime": "3s"
    }'
```

Job

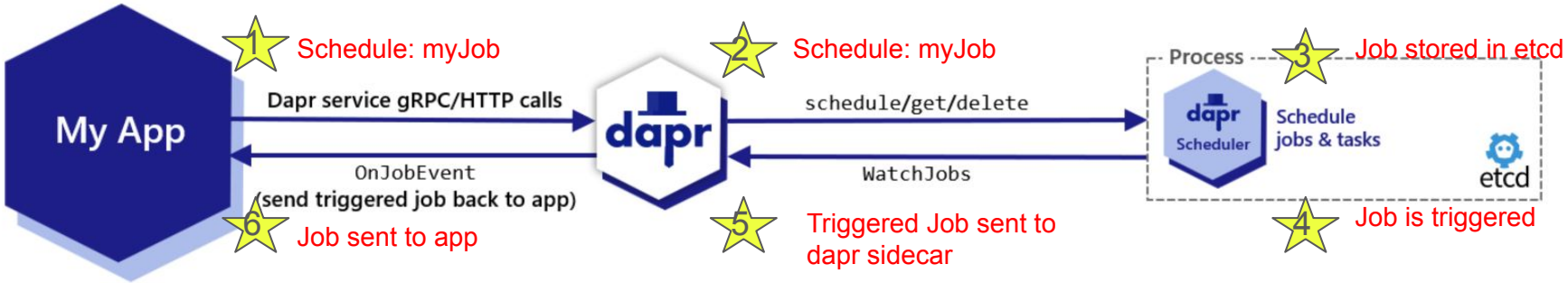

```
message Job {
  optional string schedule = 1;
  optional uint32 repeats = 2;
  optional string due_time = 3;
  optional string ttl = 4;
  google.protobuf.Any data = 5;
  optional FailurePolicy failure_policy = 6;
}
```

Jobs API


My App
Dapr service gRPC/HTTP calls
dapr
OnJobEvent
(send triggered job back to app)

Jobs API + Scheduler Service

- Schedule jobs to be executed at some point in the future



Future

- Delayed PubSub
- Scheduled Service Invocation
- Auto Scale Scheduler
- Optionally Storing Job Data Separately
- CRD job creation

Thank you for your contributions to Dapr Scheduler


Cassie


Elena


Josh


Artur


Yaron