distributed application runtime

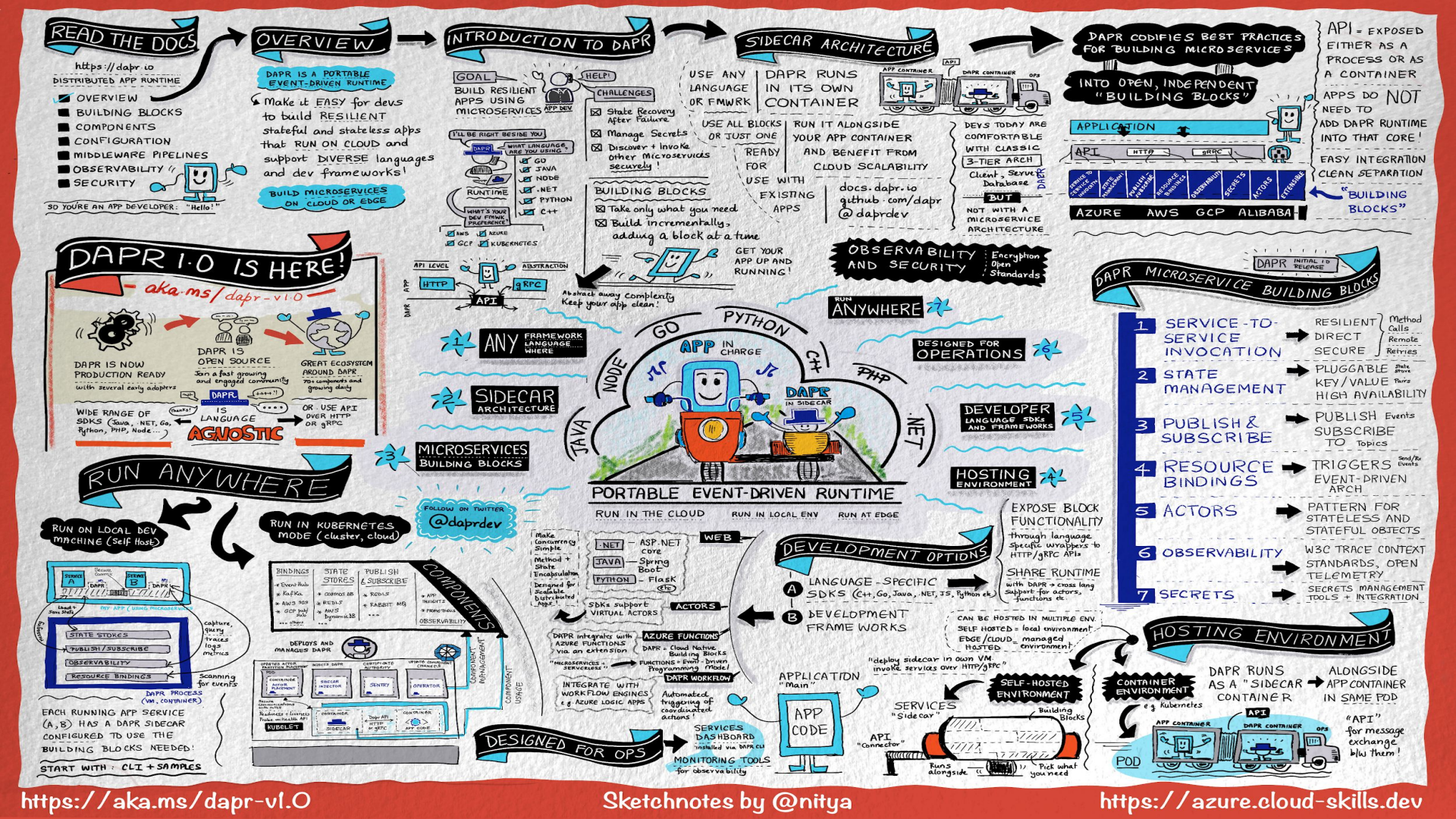
an intro

TL;DR;

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

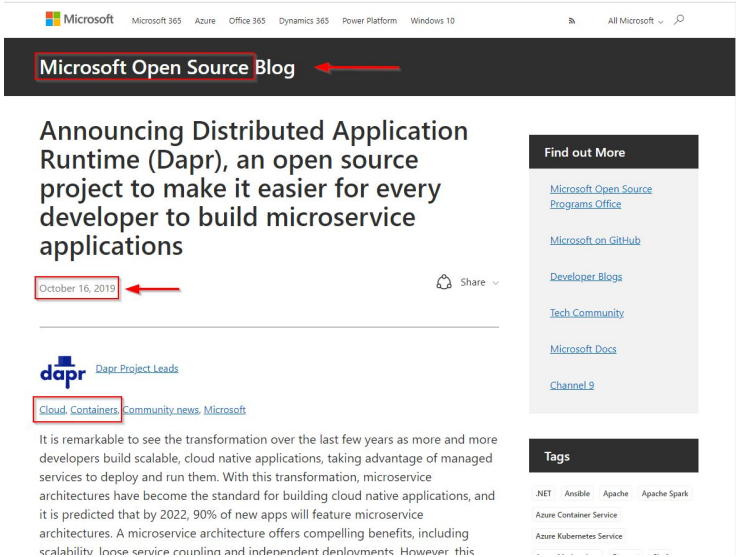
> Dapr codifies the **best practices for building microservice applications** into open, independent, **building blocks** that enable you to build portable applications with the language and framework of your choice.

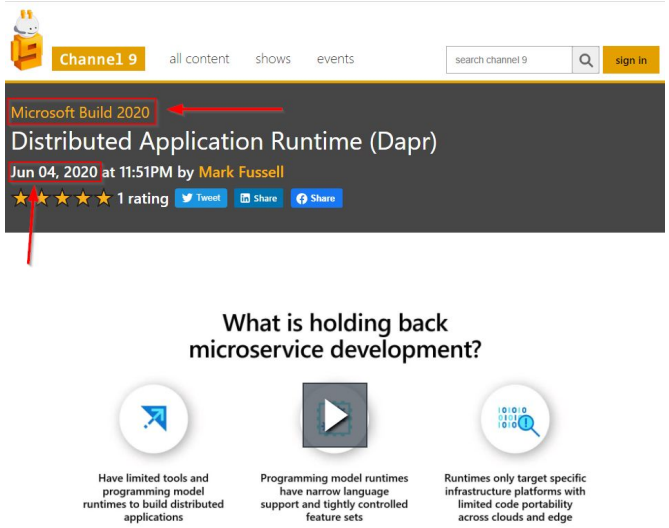
https://docs.dapr.io/concepts/overview/


READ THE DOCS
https://dapr.io
DISTRIBUTED APP RUNTIME
OVERVIEW
BUILDING BLOCKS
COMPONENTS
CONFIGURATION
MIDDLEWARE PIPELINES
OBSERVABILITY
SECURITY
SO YOU'RE AN APP DEVELOPER: "Hello!"
OVERVIEW
DAPR IS A PORTABLE EVENT-DRIVEN RUNTIME
Make it EASY for devs to build RESILIENT stateful and stateless apps that RUN ON CLOUD and support DIVERSE languages and dev frameworks!
BUILD MICROSERVICES ON CLOUD OR EDGE
INTRODUCTION TO DAPR
GOAL
BUILD RESILIENT APPS USING MICROSERVICES
APP DEV
HELP!
CHALLENGES
State Recovery After Failure
Manage Secrets
Discover + Invoke other Microservices securely
I'LL BE RIGHT BESIDE YOU
WHAT LANGUAGE, ARE YOU USING
GO
JAVA
NODE
.NET
PYTHON
C++
RUNTIME
WHAT'S YOUR DEV FRMWK PREFERENCE?
AWS
AZURE
GCP
KUBERNETES
BUILDING BLOCKS
Take only what you need
Build incrementally, adding a block at a time
GET YOUR APP UP AND RUNNING!
Abstract away Complexity Keep your app clean!
API LEVEL
ABSTRACTION
APP
DAPR
HTTP
gRPC
API
SIDECAR ARCHITECTURE
USE ANY LANGUAGE OR FMWRK
USE ALL BLOCKS OR JUST ONE
READY FOR USE WITH EXISTING APPS
DAPR RUNS IN ITS OWN CONTAINER
RUN IT ALONGSIDE YOUR APP CONTAINER AND BENEFIT FROM CLOUD SCALABILITY
docs.dapr.io
github.com/dapr
@daprdev
APP CONTAINER
API
DAPR CONTAINER
OPS
DEVS TODAY ARE COMFORTABLE WITH CLASSIC 3-TIER ARCH
Client, Server, Database
BUT
NOT WITH A MICROSERVICE ARCHITECTURE
DAPR CODIFIES BEST PRACTICES FOR BUILDING MICROSERVICES
INTO OPEN, INDEPENDENT "BUILDING BLOCKS"
APPLICATION
API
HTTP
gRPC
SERVICE TO SERVICE INVOCATION
PUB/SUB
EVENT TRIGGERS
VIRTUAL ACTORS
RESOURCE BINDINGS
OBSERVABILITY
SECRETS
ACTORS
EXTENSIBLE
DAPR
AZURE AWS GCP ALIBABA
"BUILDING BLOCKS"
API = EXPOSED EITHER AS A PROCESS OR AS A CONTAINER
APPS DO NOT NEED TO ADD DAPR RUNTIME INTO THAT CORE!
EASY INTEGRATION
CLEAN SEPARATION
OBSERVABILITY AND SECURITY
Encryption
Open Standards
DAPR 1.0 IS HERE!
aka.ms/dapr-v1.0
DAPR IS NOW PRODUCTION READY
with several early adopters
DAPR IS OPEN SOURCE
Join a fast growing and engaged community
GREAT ECOSYSTEM AROUND DAPR
70+ components and growing daily
WIDE RANGE OF SDKS (Java, .NET, Go, Python, PHP, Node...)
DAPR IS LANGUAGE AGNOSTIC
OR USE API OVER HTTP OR gRPC
DAPR MICROSERVICE BUILDING BLOCKS
DAPR INITIAL 1.0 RELEASE
1 SERVICE-TO-SERVICE INVOCATION → RESILIENT DIRECT SECURE (Method Calls, Remote, Retries)
2 STATE MANAGEMENT → PLUGGABLE (State stores) KEY/VALUE Pairs HIGH AVAILABILITY
3 PUBLISH & SUBSCRIBE → PUBLISH Events SUBSCRIBE TO Topics
4 RESOURCE BINDINGS → TRIGGERS (Send/Rx Events) EVENT-DRIVEN ARCH
5 ACTORS → PATTERN FOR STATELESS AND STATEFUL OBJECTS
6 OBSERVABILITY → W3C TRACE CONTEXT STANDARDS, OPEN TELEMETRY
7 SECRETS → SECRETS MANAGEMENT TOOLS + INTEGRATION
1 ANY FRAMEWORK LANGUAGE WHERE
2 SIDECAR ARCHITECTURE
3 MICROSERVICES BUILDING BLOCKS
4 HOSTING ENVIRONMENT
5 DEVELOPER LANGUAGE SDKs AND FRAMEWORKS
6 DESIGNED FOR OPERATIONS
RUN ANYWHERE
PORTABLE EVENT-DRIVEN RUNTIME
APP IN CHARGE
DAPR IN SIDECAR
GO
PYTHON
C++
PHP
.NET
JAVA
NODE
RUN IN THE CLOUD
RUN IN LOCAL ENV
RUN AT EDGE
RUN ANYWHERE
RUN ON LOCAL DEV MACHINE (Self Host)
RUN IN KUBERNETES MODE (cluster, cloud)
FOLLOW ON TWITTER @daprdev
SERVICE A
SERVICE B
DAPR
MY APP (USING MICROSERVICES)
Local + Java Shells
Kubernetes
STATE STORES
PUBLISH/SUBSCRIBE
OBSERVABILITY
RESOURCE BINDINGS
capture, query traces logs metrics
Scanning for events
DAPR PROCESS (VM, CONTAINER)
EACH RUNNING APP SERVICE (A, B) HAS A DAPR SIDECAR CONFIGURED TO USE THE BUILDING BLOCKS NEEDED!
START WITH: CLI + SAMPLES
COMPONENTS
BINDINGS
Event Hubs
Kafka
AWS SNS
GCP pub/sub
... others
STATE STORES
Cosmos DB
Redis
AWS DynamoDB
...
PUBLISH & SUBSCRIBE
Redis
RabbitMQ
...
APP INSIGHTS
Prometheus
OBSERVABILITY
DEPLOYS AND MANAGES DAPR
SIDECAR INJECTOR
SENTRY
OPERATOR
KUBELET
CONTAINER
SIDECAR
Dapr API HTTP or gRPC
APP CODE
COMPONENT MANAGEMENT
COMPONENT USAGE
DEVELOPMENT OPTIONS
A LANGUAGE-SPECIFIC SDKs (C++, Go, Java, .NET, JS, Python etc)
B DEVELOPMENT FRAME WORKS
EXPOSE BLOCK FUNCTIONALITY through language specific wrappers to HTTP/gRPC APIs
SHARE RUNTIME with DAPR → cross lang support for actors, functions etc.
WEB
.NET — ASP.NET Core
JAVA — Spring Boot
PYTHON — Flask (etc)
Make concurrency Simple
Method + State Encapsulation
Designed for Scalable, Distributed Apps
ACTORS
SDKs support VIRTUAL ACTORS
AZURE FUNCTIONS
DAPR integrates with AZURE FUNCTIONS via an extension
DAPR = Cloud Native Building Blocks
FUNCTIONS = Event-Driven Programming Model
"MICROSERVICES + SERVERLESS"
DAPR WORKFLOW
INTEGRATE WITH WORKFLOW ENGINES e.g. AZURE LOGIC APPS
Automated triggering of coordinated actions
DESIGNED FOR OPS
SERVICES DASHBOARD installed via DAPR CLI
MONITORING TOOLS for observability
APPLICATION "Main"
APP CODE
SERVICES "Sidecar"
API "Connector"
Runs alongside
Building Blocks
Pick what you need
"deploy sidecar in own VM invoke services over HTTP/gRPC"
HOSTING ENVIRONMENT
CAN BE HOSTED IN MULTIPLE ENV.
SELF HOSTED = local environment
EDGE/CLOUD HOSTED = managed environment
SELF-HOSTED ENVIRONMENT
CONTAINER ENVIRONMENT
e.g. Kubernetes
DAPR RUNS AS A "SIDECAR" CONTAINER
ALONGSIDE APP CONTAINER IN SAME POD
APP CONTAINER
API
DAPR CONTAINER
OPS
POD
"API" for message exchange b/w them!

History

- originated within Microsoft
- announced **2019**, broader audience **Microsoft Build 2020**
  - IMHO: possibly because nobody wanted to use Service Fabric?

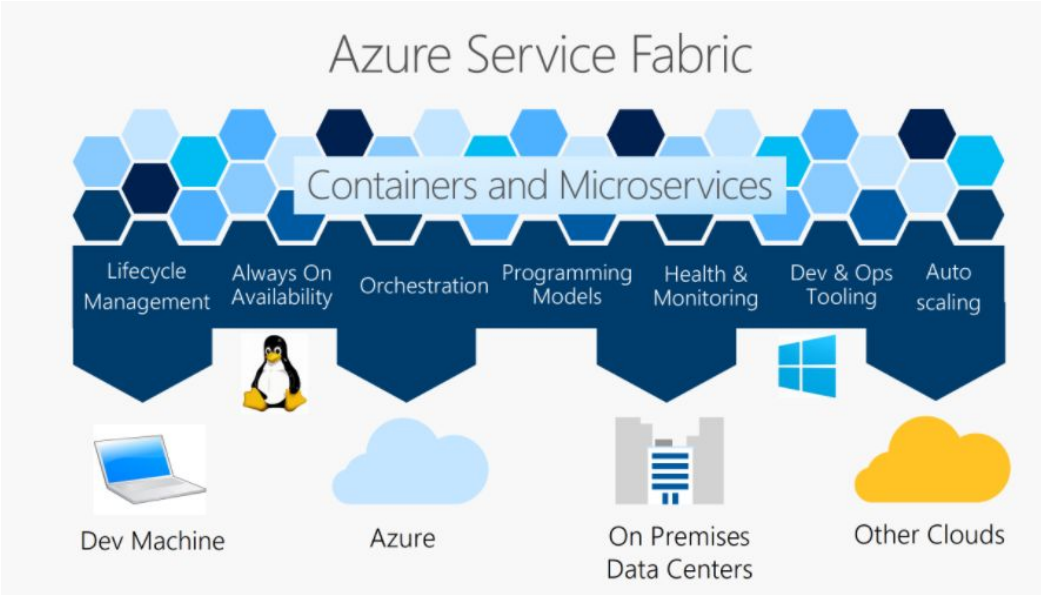




(detour) Service Fabric

- Microsoft-specific technology not really widely used

old Microsoft! proprietary



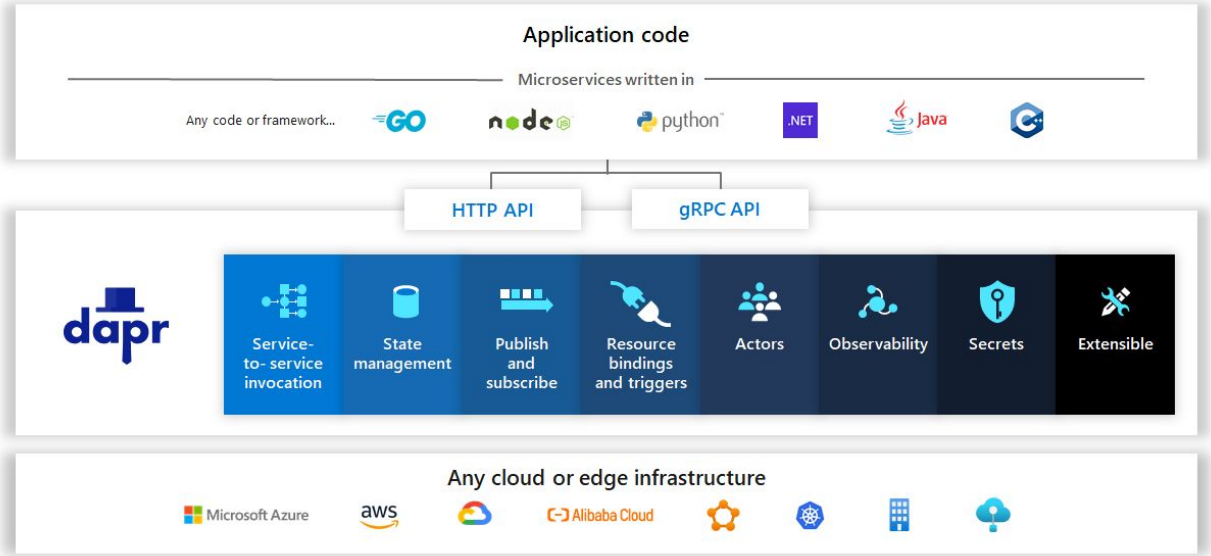

```xml
<?xml version="1.0" encoding="utf-8"?>
<ApplicationManifest xmlns:xsd="https://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchem
  <Parameters>
    <Parameter Name="VotingData_MinReplicaSetSize" DefaultValue="3" />
    <Parameter Name="VotingData_PartitionCount" DefaultValue="1" />
    <Parameter Name="VotingData_TargetReplicaSetSize" DefaultValue="3" />
    <Parameter Name="VotingWeb_InstanceCount" DefaultValue="-1" />
  </Parameters>
  <!-- Import the ServiceManifest from the ServicePackage. The ServiceManifestName and ServiceManifestVersion
       should match the Name and Version attributes of the ServiceManifest element defined in the
       ServiceManifest.xml file. -->
  <ServiceManifestImport>
    <ServiceManifestRef ServiceManifestName="VotingDataPkg" ServiceManifestVersion="1.0.0" />
    <ConfigOverrides />
  </ServiceManifestImport>
  <ServiceManifestImport>
    <ServiceManifestRef ServiceManifestName="VotingWebPkg" ServiceManifestVersion="1.0.0" />
    <ConfigOverrides />
  </ServiceManifestImport>
  <DefaultServices>
    <!-- The section below creates instances of service types, when an instance of this
         application type is created. You can also create one or more instances of service type using the
         ServiceFabric PowerShell module.

         The attribute ServiceTypeName below must match the name defined in the imported ServiceManifest.xml f
    <Service Name="VotingData">
      <StatefulService ServiceTypeName="VotingDataType" TargetReplicaSetSize="[VotingData_TargetReplicaSetSize
        <UniformInt64Partition PartitionCount="[VotingData_PartitionCount]" LowKey="0" HighKey="25" />
      </StatefulService>
    </Service>
    <Service Name="VotingWeb" ServicePackageActivationMode="ExclusiveProcess">
      <StatelessService ServiceTypeName="VotingWebType" InstanceCount="[VotingWeb_InstanceCount]">
        <SingletonPartition />
        <PlacementConstraints>(NodeType==NodeType0)</PlacementConstraints
      </StatelessService>
    </Service>
  </DefaultServices>
</ApplicationManifest>
```

https://docs.microsoft.com/en-us/azure/service-fabric/

Overview

- **Building-Blocks** to somehow standardize/simplify cloud development
- No custom SDKs **only HTTP/gRPC (*)**
- Can be used with **any language** and with **multiple environments** (systems/cloud providers)



Building Blocks

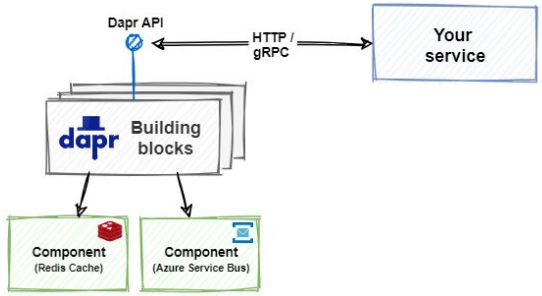
- **Typical components/services** needed for app development
  - https://docs.dapr.io/reference/components-reference/
  - “young project” - many components Alpha/Beta status!
  - cannot hide Microsoft Background (Azure components GA)
- Simple use/integration because of **HTTP/gRPC**
- Hide cloud platform-/technology-details - **abstraction layer**

SDK languages

Language	Status	Client SDK	Server extensions	Actor SDK
.NET	Stable	✔	ASP.NET Core	✔
Python	Stable	✔	gRPC	FastAPI Flask
Java	Stable	✔	Spring Boot	✔
Go	Stable	✔	✔	
PHP	Stable	✔	✔	✔
C++	In development	✔		
Rust	In development	✔		
Javascript	In development	✔		

Implementation

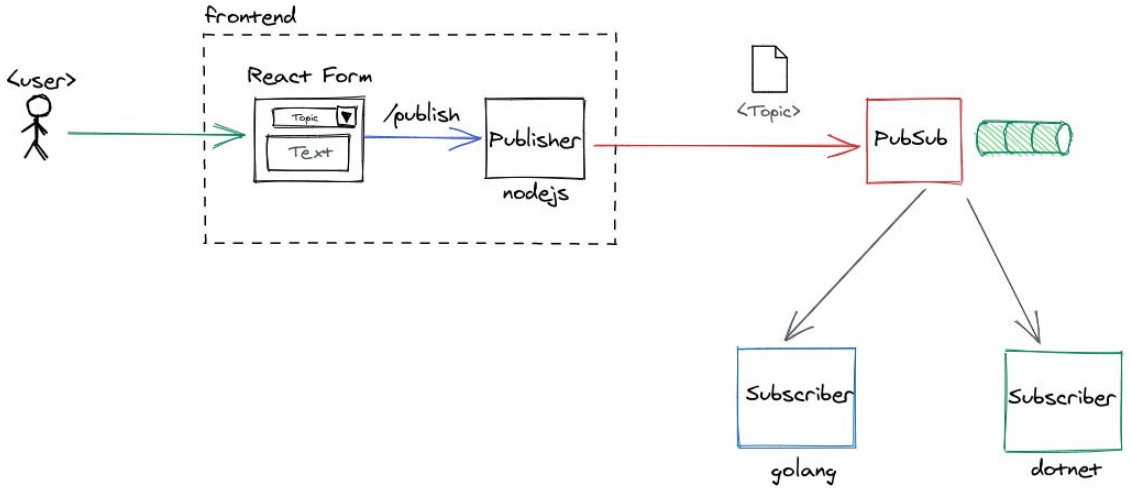
- Development in the open https://github.com/dapr/dapr / Community Calls - YouTube
- v1.2.0 current, MIT License, golang based (❤)
- Dapr logic/building-blocks implemented as sidecars
- Expose HTTP/gRPC API for invocation (dapr API / dapr GRPC)
- Dapr sidecar takes care to interact with other services/components (abstraction)



- Standalone deployment (local development, cli)
- k8s deployment (quickstarts/hello-kubernetes at master · dapr/quickstarts (github.com))

DEMO

- Dev example with simple pubsub logic
- dapr CLI
- k8s with dapr (use redis)
- k8s with dapr (use azure service-bus)



DEMO

Dev example - PupSub

- **expressjs** -> simple POST to http://localhost:3500/**v1.0/publish**/**pubsubname**/**<TOPIC>**

```
const daprPort = process.env.DAPR_HTTP_PORT || 3500;
const daprUrl = `http://localhost:${daprPort}/v1.0`;
const port = 8080;
const pubsubName = 'pubsub';

app.post('/publish', (req, res) => {
  console.log("Publishing: ", req.body);
  const publishUrl = `${daprUrl}/publish/${pubsubName}/${req.body.messageType}`;
  request( { uri: publishUrl, method: 'POST', json: req.body } );
  res.sendStatus(200);
});
```

- **golang** -> register/consume (**/dapr/subscribe**) by HTTP-handler (* programmatic-subscription)

```
func subscribe() http.HandlerFunc {
    return func(w http.ResponseWriter, r *http.Request) {
        sub := []Subscribe{
            {
                PubSubName: pubSubName,
                Topic:      "ALL",
                Route:      "receive_all",
            },
            {
                PubSubName: pubSubName,
                Topic:      "Topic1",
                Route:      "receive_b",
            },
        }
```

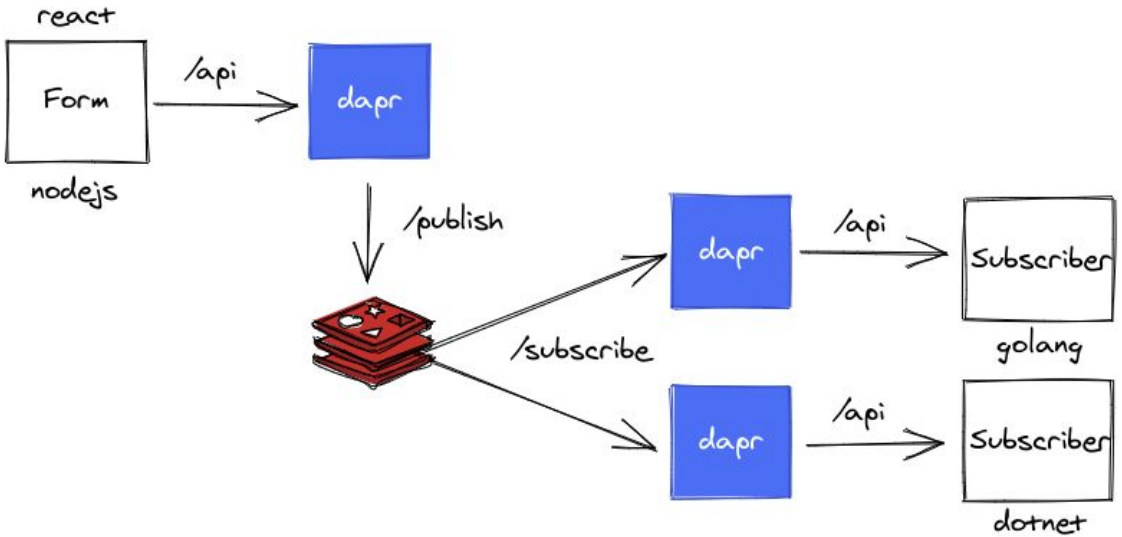
```
func procMessage(route string) http.HandlerFunc {
    return func(w http.ResponseWriter, r *http.Request) {
        msg, err := getMessage(r.Body)
        defer r.Body.Close()
        if err != nil {
            http.Error(w, err.Error(), 500)
            return
        }
        log.Printf("📜 via '%s', for '%s' with message '%s'", route, msg.Topic, msg.Data.Message)
        w.WriteHeader(http.StatusOK)
    }
}
```

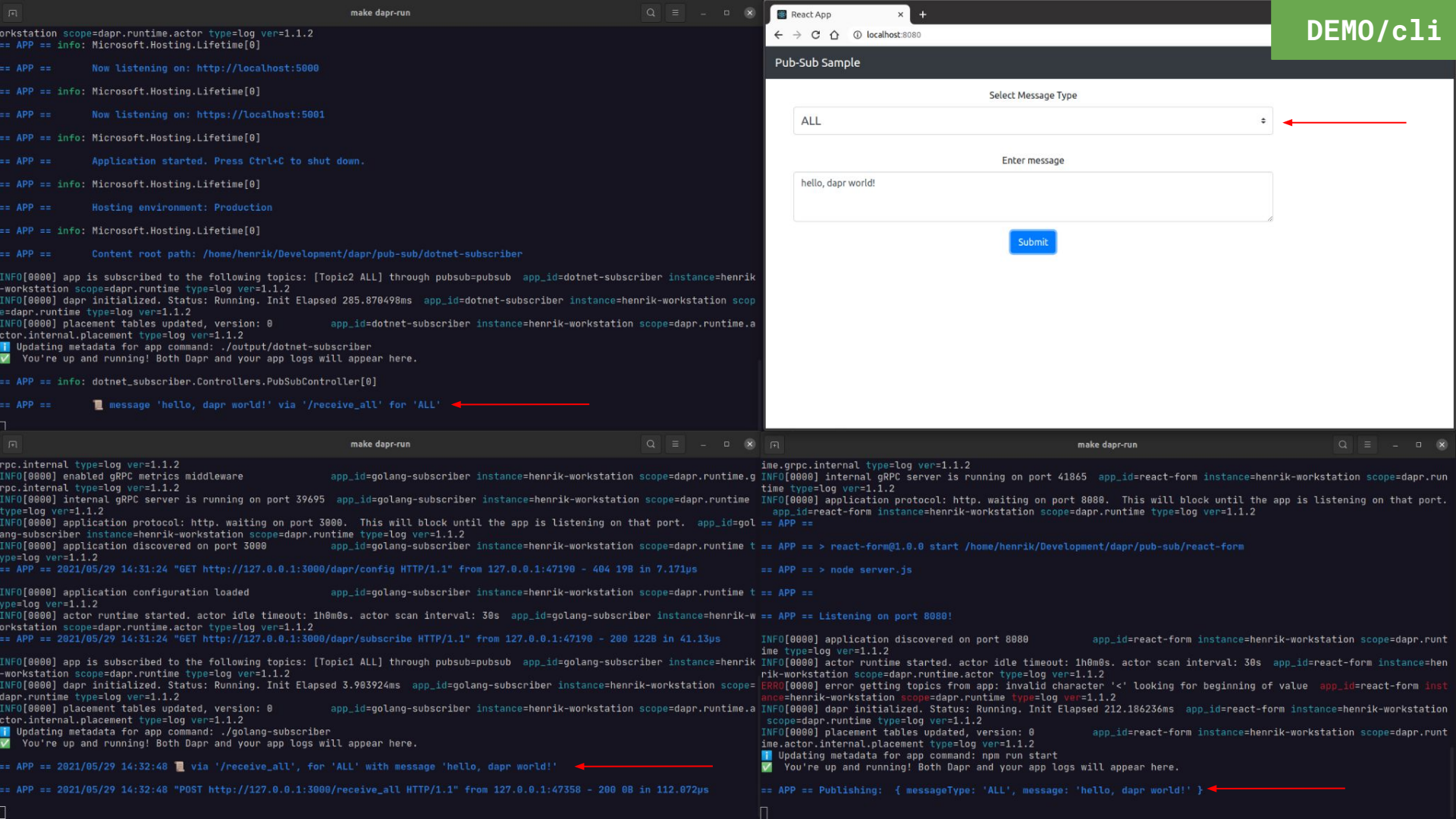
src: https://github.com/bihe/dapr-intro

DEMO/cli

Dev example - PupSub

- dapr run --app-id **golang-subscriber** --app-port 3000 ./golang-subscriber
- dapr run --app-id **dotnet-subscriber** --app-port 5000 ./output/dotnet-subscriber
- dapr run --app-id **react-form** --app-port 8080 npm run start



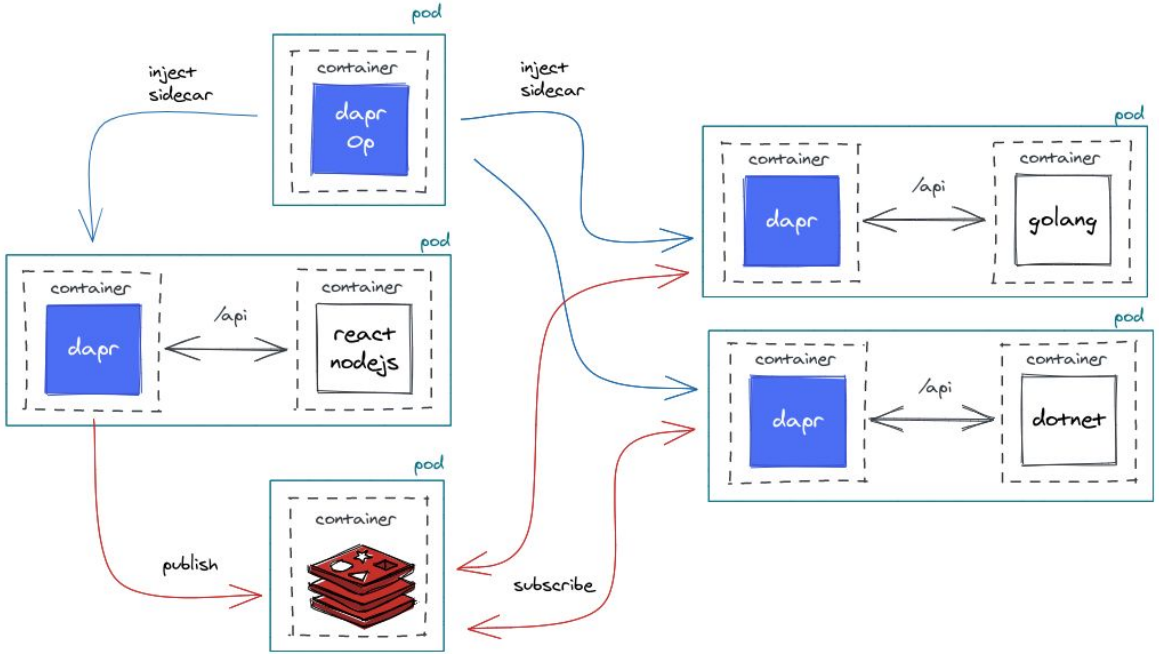

make dapr-run
orkstation scope=dapr.runtime.actor type=log ver=1.1.2
== APP == info: Microsoft.Hosting.Lifetime[0]
== APP ==   Now listening on: http://localhost:5000
== APP == info: Microsoft.Hosting.Lifetime[0]
== APP ==   Now listening on: https://localhost:5001
== APP == info: Microsoft.Hosting.Lifetime[0]
== APP ==   Application started. Press Ctrl+C to shut down.
== APP == info: Microsoft.Hosting.Lifetime[0]
== APP ==   Hosting environment: Production
== APP == info: Microsoft.Hosting.Lifetime[0]
== APP ==   Content root path: /home/henrik/Development/dapr/pub-sub/dotnet-subscriber
INFO[0000] app is subscribed to the following topics: [Topic2 ALL] through pubsub=pubsub app_id=dotnet-subscriber instance=henrik
-workstation scope=dapr.runtime type=log ver=1.1.2
INFO[0000] dapr initialized. Status: Running. Init Elapsed 285.870498ms app_id=dotnet-subscriber instance=henrik-workstation scop
e=dapr.runtime type=log ver=1.1.2
INFO[0000] placement tables updated, version: 0 app_id=dotnet-subscriber instance=henrik-workstation scope=dapr.runtime.a
ctor.internal.placement type=log ver=1.1.2
ℹ️ Updating metadata for app command: ./output/dotnet-subscriber
✅ You're up and running! Both Dapr and your app logs will appear here.
== APP == info: dotnet_subscriber.Controllers.PubSubController[0]
== APP ==   📜 message 'hello, dapr world!' via '/receive_all' for 'ALL'
React App
localhost:8080
Pub-Sub Sample
Select Message Type
ALL
Enter message
hello, dapr world!
Submit
DEMO/cli
make dapr-run
rpc.internal type=log ver=1.1.2
INFO[0000] enabled gRPC metrics middleware app_id=golang-subscriber instance=henrik-workstation scope=dapr.runtime.g
rpc.internal type=log ver=1.1.2
INFO[0000] internal gRPC server is running on port 39695 app_id=golang-subscriber instance=henrik-workstation scope=dapr.runtime
type=log ver=1.1.2
INFO[0000] application protocol: http. waiting on port 3000. This will block until the app is listening on that port. app_id=gol
ang-subscriber instance=henrik-workstation scope=dapr.runtime type=log ver=1.1.2
INFO[0000] application discovered on port 3000 app_id=golang-subscriber instance=henrik-workstation scope=dapr.runtime t
ype=log ver=1.1.2
== APP == 2021/05/29 14:31:24 "GET http://127.0.0.1:3000/dapr/config HTTP/1.1" from 127.0.0.1:47190 - 404 19B in 7.171µs
INFO[0000] application configuration loaded app_id=golang-subscriber instance=henrik-workstation scope=dapr.runtime t
ype=log ver=1.1.2
INFO[0000] actor runtime started. actor idle timeout: 1h0m0s. actor scan interval: 30s app_id=golang-subscriber instance=henrik-w
orkstation scope=dapr.runtime.actor type=log ver=1.1.2
== APP == 2021/05/29 14:31:24 "GET http://127.0.0.1:3000/dapr/subscribe HTTP/1.1" from 127.0.0.1:47190 - 200 122B in 41.13µs
INFO[0000] app is subscribed to the following topics: [Topic1 ALL] through pubsub=pubsub app_id=golang-subscriber instance=henrik
-workstation scope=dapr.runtime type=log ver=1.1.2
INFO[0000] dapr initialized. Status: Running. Init Elapsed 3.903924ms app_id=golang-subscriber instance=henrik-workstation scope=
dapr.runtime type=log ver=1.1.2
INFO[0000] placement tables updated, version: 0 app_id=golang-subscriber instance=henrik-workstation scope=dapr.runtime.a
ctor.internal.placement type=log ver=1.1.2
ℹ️ Updating metadata for app command: ./golang-subscriber
✅ You're up and running! Both Dapr and your app logs will appear here.
== APP == 2021/05/29 14:32:48 📜 via '/receive_all', for 'ALL' with message 'hello, dapr world!'
== APP == 2021/05/29 14:32:48 "POST http://127.0.0.1:3008/receive_all HTTP/1.1" from 127.0.0.1:47358 - 200 0B in 112.072µs
make dapr-run
ime.grpc.internal type=log ver=1.1.2
INFO[0000] internal gRPC server is running on port 41865 app_id=react-form instance=henrik-workstation scope=dapr.run
time type=log ver=1.1.2
INFO[0000] application protocol: http. waiting on port 8080. This will block until the app is listening on that port.
app_id=react-form instance=henrik-workstation scope=dapr.runtime type=log ver=1.1.2
== APP ==
== APP == > react-form@1.0.0 start /home/henrik/Development/dapr/pub-sub/react-form
== APP == > node server.js
== APP ==
== APP == Listening on port 8080!
INFO[0000] application discovered on port 8080 app_id=react-form instance=henrik-workstation scope=dapr.runt
ime type=log ver=1.1.2
INFO[0000] actor runtime started. actor idle timeout: 1h0m0s. actor scan interval: 30s app_id=react-form instance=hen
rik-workstation scope=dapr.runtime.actor type=log ver=1.1.2
ERRO[0000] error getting topics from app: invalid character '<' looking for beginning of value app_id=react-form inst
ance=henrik-workstation scope=dapr.runtime type=log ver=1.1.2
INFO[0000] dapr initialized. Status: Running. Init Elapsed 212.186236ms app_id=react-form instance=henrik-workstation
scope=dapr.runtime type=log ver=1.1.2
INFO[0000] placement tables updated, version: 0 app_id=react-form instance=henrik-workstation scope=dapr.runt
ime.actor.internal.placement type=log ver=1.1.2
ℹ️ Updating metadata for app command: npm run start
✅ You're up and running! Both Dapr and your app logs will appear here.
== APP == Publishing: { messageType: 'ALL', message: 'hello, dapr world!' }

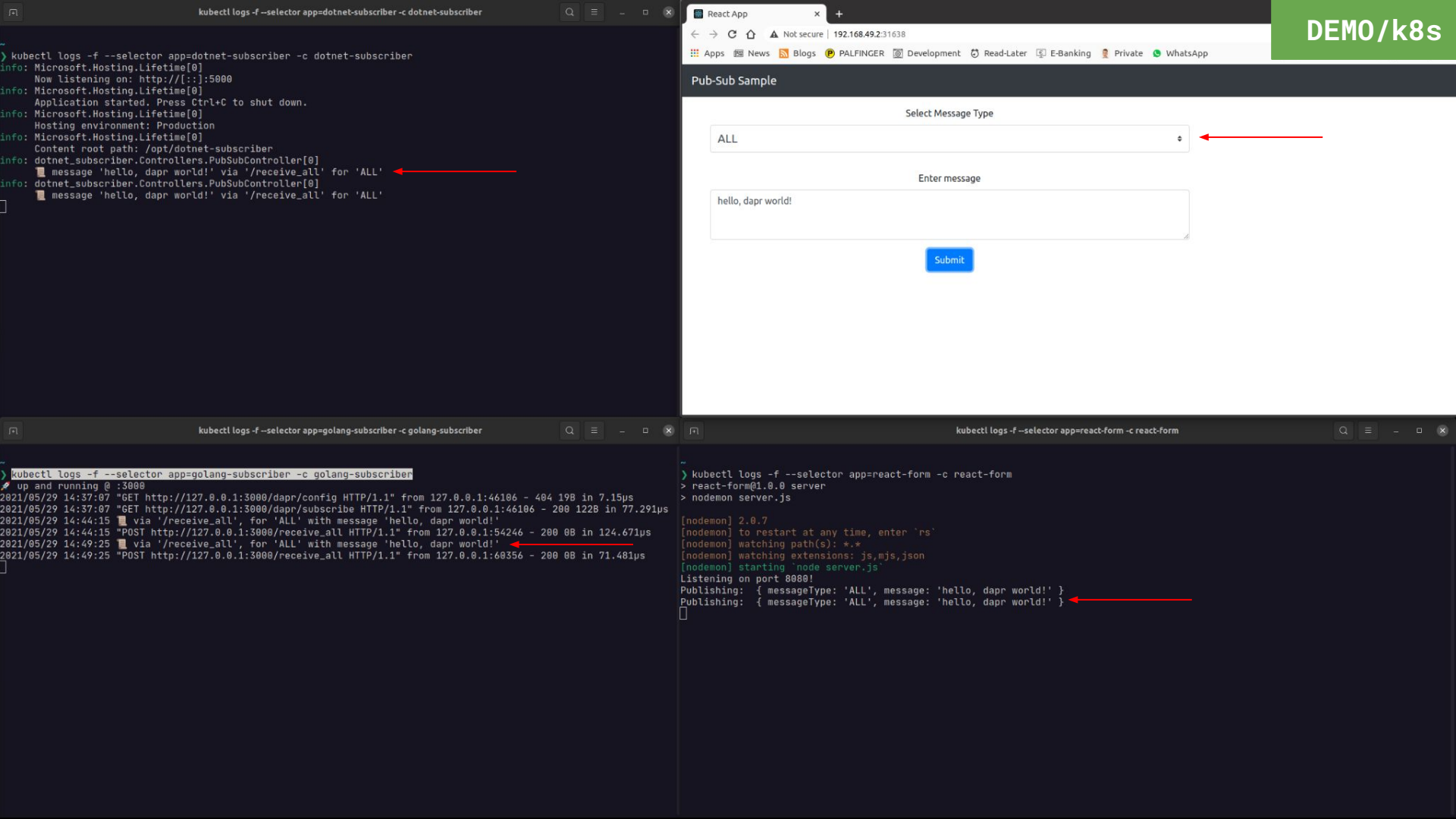
DEMO/k8s

Dev example - PupSub @ k8s

- k8s deployment with dapr sidecars

```yaml
apiVersion: dapr.io/v1alpha1
kind: Component
metadata:
  name: pubsub
spec:
  type: pubsub.redis
  version: v1
  metadata:
  - name: redisHost
    value: redis-master:6379
  - name: redisPassword
    secretKeyRef:
      name: redis
      key: redis-password
auth:
  secretStore: kubernetes
```



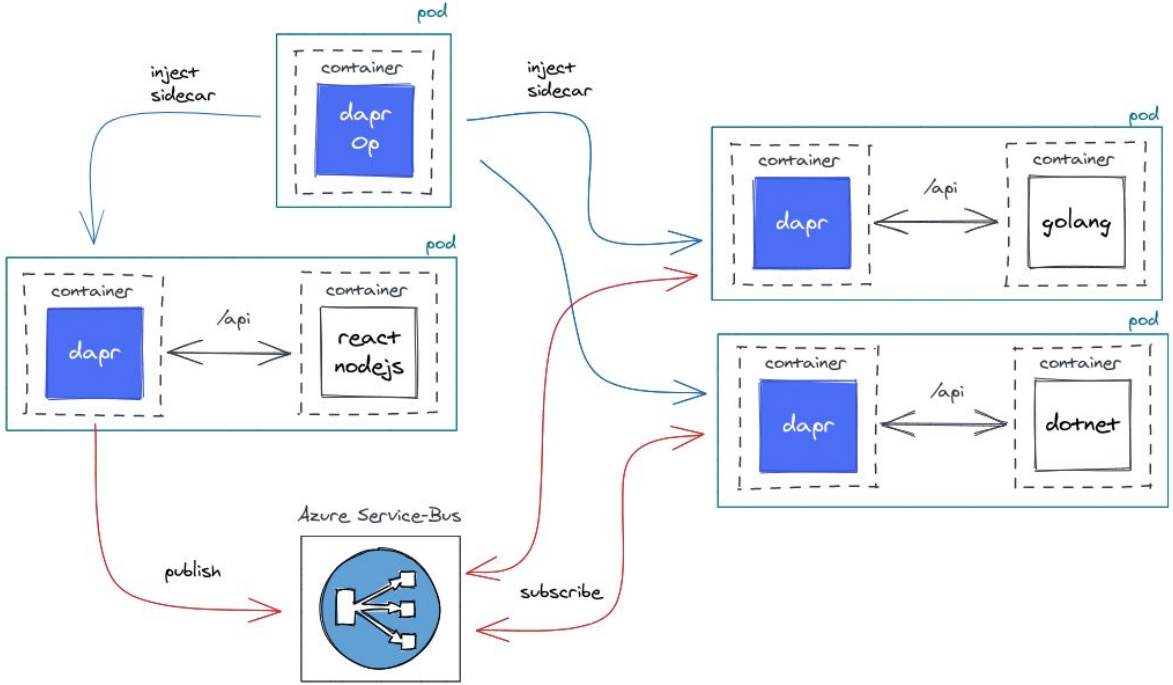

kubectl logs -f --selector app=dotnet-subscriber -c dotnet-subscriber
❯ kubectl logs -f --selector app=dotnet-subscriber -c dotnet-subscriber
info: Microsoft.Hosting.Lifetime[0]
Now listening on: http://[::]:5000
info: Microsoft.Hosting.Lifetime[0]
Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
Hosting environment: Production
info: Microsoft.Hosting.Lifetime[0]
Content root path: /opt/dotnet-subscriber
info: dotnet_subscriber.Controllers.PubSubController[0]
message 'hello, dapr world!' via '/receive_all' for 'ALL'
info: dotnet_subscriber.Controllers.PubSubController[0]
message 'hello, dapr world!' via '/receive_all' for 'ALL'
React App
Not secure | 192.168.49.2:31638
Apps News Blogs PALFINGER Development Read-Later E-Banking Private WhatsApp
DEMO/k8s
Pub-Sub Sample
Select Message Type
ALL
Enter message
hello, dapr world!
Submit
kubectl logs -f --selector app=golang-subscriber -c golang-subscriber
❯ kubectl logs -f --selector app=golang-subscriber -c golang-subscriber
up and running @ :3000
2021/05/29 14:37:07 "GET http://127.0.0.1:3000/dapr/config HTTP/1.1" from 127.0.0.1:46106 - 404 19B in 7.15µs
2021/05/29 14:37:07 "GET http://127.0.0.1:3000/dapr/subscribe HTTP/1.1" from 127.0.0.1:46106 - 200 122B in 77.291µs
2021/05/29 14:44:15 via '/receive_all', for 'ALL' with message 'hello, dapr world!'
2021/05/29 14:44:15 "POST http://127.0.0.1:3000/receive_all HTTP/1.1" from 127.0.0.1:54246 - 200 0B in 124.671µs
2021/05/29 14:49:25 via '/receive_all', for 'ALL' with message 'hello, dapr world!'
2021/05/29 14:49:25 "POST http://127.0.0.1:3000/receive_all HTTP/1.1" from 127.0.0.1:60356 - 200 0B in 71.481µs
kubectl logs -f --selector app=react-form -c react-form
❯ kubectl logs -f --selector app=react-form -c react-form
> react-form@1.0.0 server
> nodemon server.js
[nodemon] 2.0.7
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node server.js`
Listening on port 8080!
Publishing: { messageType: 'ALL', message: 'hello, dapr world!' }
Publishing: { messageType: 'ALL', message: 'hello, dapr world!' }

DEMO/k8s

Dev example - PupSub @ k8s

- swap redis with Azure Service-Bus

```yaml
apiVersion: dapr.io/v1alpha1
kind: Component
metadata:
  name: pubsub
spec:
  type: pubsub.azure.servicebus
  version: v1
  metadata:
  - name: connectionString # Require
    secretKeyRef:
      name: az-sb
      key: connstr
auth:
  secretStore: kubernetes
```



Links

- https://docs.dapr.io/**getting-started**/
- https://github.com/dapr/**quickstarts**/tree/master/pub-sub
- https://docs.dapr.io/developing-applications/building-blocks/pubsub/**pubsub-overview**/
- https://docs.dapr.io/reference/components-reference/supported-pubsub/**setup-azure-servicebus**/
- PubSub with Azure Service Bus: https://www.youtube.com/watch?v=umrUlfrZqKk
- https://github.com/bihe/**dapr-intro**
- https://blog.dapr.io/posts/2021/03/02/**a-visual-guide-to-dapr**/