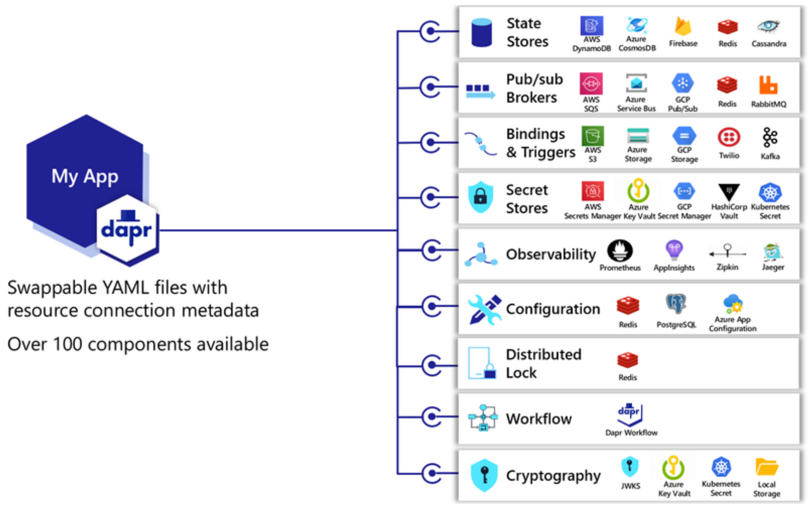
Zero Trust Security for Distributed Applications with Dapr

What is Dapr ?

Dapr (Distributed Application Runtime) project. Dapr is a new way to build modern distributed applications. Born at Microsoft in 2019 as an incubations project, it was donated to the Cloud Native Computing Foundation in 2021 and is now a highly successful **open-source project**. Dapr helps you write flexible and secure microservices that leverage industry proven best practices to build connected distributed applications faster.

By letting Dapr's sidecar take care of the complex challenges such as service discovery, message broker integration, encryption, observability, and secret management, you can focus on business logic and keep your code simple.

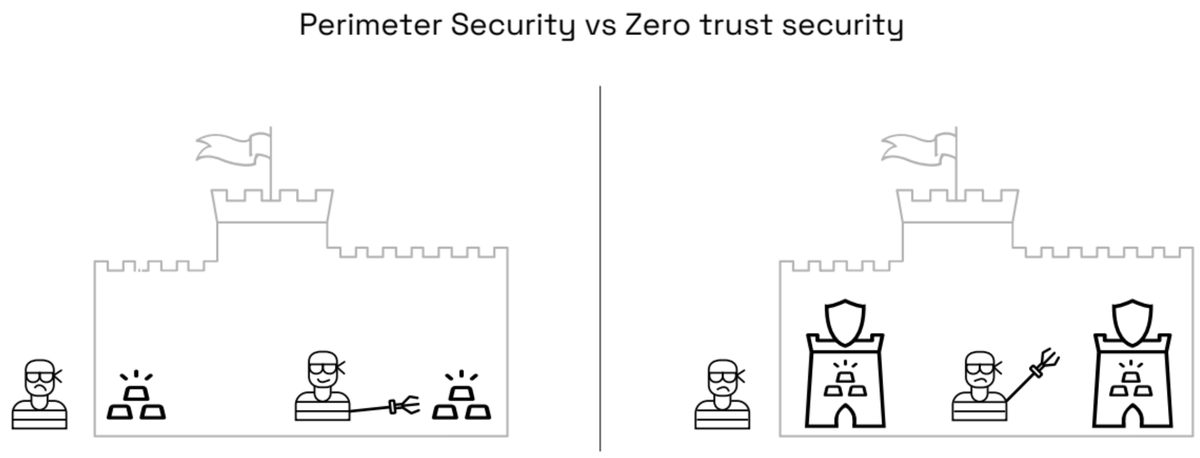


Dapr improves the zero trust security posture of distributed systems out of the box by assigning application identities to all apps, ensuring that mTLS is enabled by default for all interservice and infrastructure communication.

The standards around security in software development are ever increasing in response to the need for greater protection. This article looks at the open source project Dapr, distributed application runtime, which contains a rich security feature set that allows developers to "shift left" with security and embed industry-standard best practices into their applications during development. Dapr provides a set of APIs to solve common distributed systems challenges around state management, workflow and data.

What Is Zero Trust Security?

The U.S. Department of Defense Zero Trust Reference Architecture defines it as follows: *"No actor, system, network, or service operating outside or within the security perimeter is trusted.Instead, we must verify anything and everything attempting to establish access. It is a dramatic paradigm shift in philosophy of how we secure our infrastructure, networks and data, from verify once at the perimeter to continual verification of each user, device, application and transaction."*



Perimeter security vs. zero trust security [KubeCrash]

Zero trust security establishes the principle that nothing is trusted automatically within a system, and that trust must be proven repeatedly during every system interaction. This ensures that even if a bad actor manages to breach the system's network perimeter, they will not be able to cause significant damage, if any. This shift in security philosophy is a response to the increasing sophistication of cyber threats and the need for more robust security measures in software development.

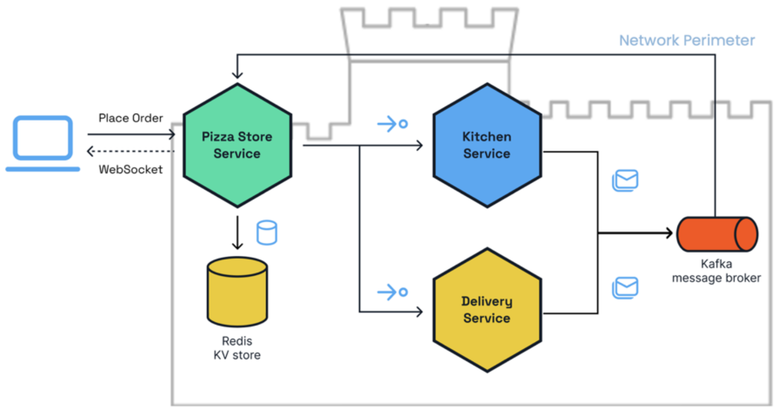
The concept of the zero trust security model has been around since the early 1990s. However, it's no longer just a buzzword. It has evolved into a security standard in software development. In fact, Gartner predicts that by 2025, 60% of companies will adopt zero trust solutions over virtual private networks as their security boundary.

A Developer Perspective

These security models are typically thought of at the infrastructure and networking level, but many parts of zero trust security and architectures are becoming developer concerns. A part of this is due to the rise in popularity of distributed systems since applications are required to communicate across the network, between one another, and to underlying infrastructure more frequently, leading to an increased number of communication channels that can be targeted for attack. Developers and engineering teams write these applications have to "shift left" security concerns to enforce security best practices earlier in the software development life cycle.

To illustrate this, let's take a sample application that we'll call the Pizza Store system. This standard event-driven architecture with multiple services communicating synchronously and asynchronously takes dependencies on various infrastructure providers such as a state store and message broker. The Pizza Store service is responsible for taking user orders and invoking the kitchen and delivery services for cooking and delivery tasks.

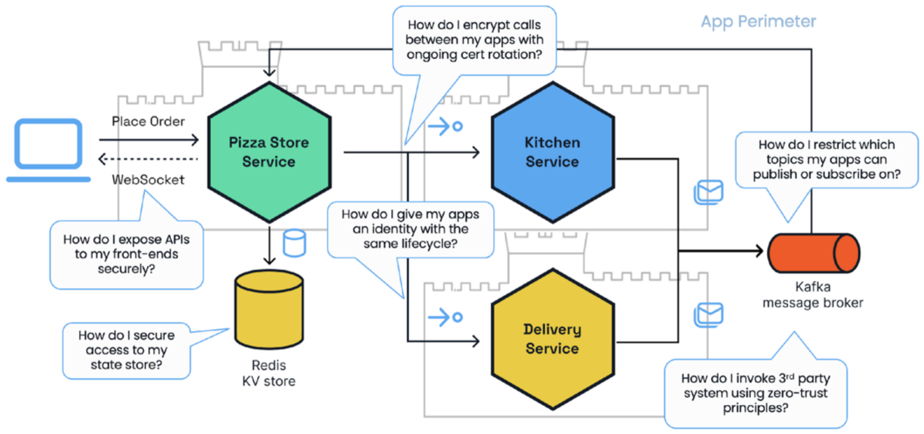
Looking at this system with a traditional perimeter security model, the trust boundary exists outside the system's network, with a single point of ingress often implemented via an API gateway or reverse proxy. The ingress is typically the point of security validation where all communication flows between system applications and infrastructure services.



Pizza Store system with a perimeter security model

Considering the zero trust security paradigm, you can no longer trust the internal system network, and an application must act as the security perimeter. By design, all communication outside the app, whether from the application to other apps, infrastructure services or end users, needs to be verified. This comes with many concerns that need to be considered in addition to the increasing number of application expectations around scale, resiliency and performance that developers are already dealing with. These concerns include:

- Establishing an application identity with the same life cycle.
- Principle least privilege infrastructure access.
- Invoking third-party systems securely.
- Encryption between services and infrastructure.
- Exposing APIs to frontends via gateway.


App Perimeter
How do I encrypt calls between my apps with ongoing cert rotation?
Place Order
WebSocket
Pizza Store Service
Kitchen Service
How do I restrict which topics my apps can publish or subscribe on?
How do I expose APIs to my front-ends securely?
How do I give my apps an identity with the same lifecycle?
Kafka message broker
Delivery Service
How do I secure access to my state store?
Redis KV store
How do I invoke 3rd party system using zero-trust principles?

Developer concerns about the Pizza Store system in a zero trust security model

Dapr Security Features

Dapr, the distributed application runtime, is a set of distributed systems APIs with built-in security features at the application level. Over 10 APIs can be used from any programming language via gRPC or HTTP to implement common distributed systems challenges and codify microservices best practices. Dapr also takes care of several cross-cutting concerns necessary for microservices, including features around distributed tracing, resiliency and security.

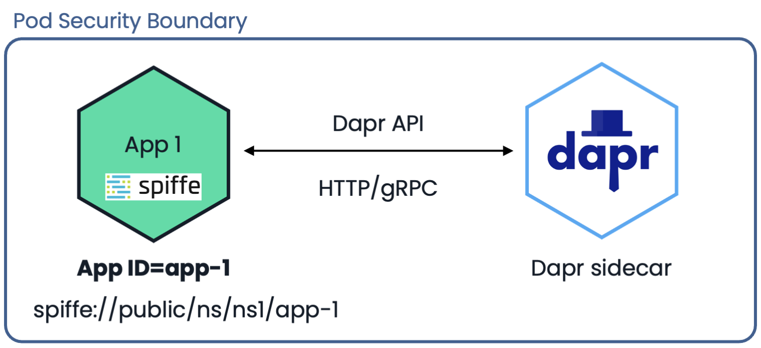
Now, we will delve into the security features of Dapr that help developers build zero trust distributed applications.

Dapr Application Identity

Since we can no longer trust the network and the application is required to act as the security perimeter, we need an intrinsic application identity to establish a new trust boundary around the service.

This is where the idea of Dapr application identities (App IDs) comes in. Dapr App IDs are:

- Strongly attested cryptographic identities using SPIFFE IDs.
- Globally unique with secrets aligned to the app’s life cycle.
- Used for authorizing connections to other apps.
- Layered with policies that can enable app and infrastructure access.



Dapr security boundary containing the application and Dapr process (sidecar)

On Kubernetes, this process is facilitated by all Dapr sidecars and control-plane services possessing a unique X.509 certificate containing the service's App ID, represented as a SPIFFE identity with the format *spiffe://<trustdomain>/ns/<namespace>/<appid>*. Applications use this certificate to attest their identity to other Dapr apps using mutual TLS (mTLS), a process known as asymmetric cryptography.

While workloads may share the same identity, as is the case when multiple replicas of the same control-plane service or application are running, each replica uses a unique, locally generated private key backing its certificate that is rotated regularly and never leaves the processes memory. Each service requests its identity certificate from the Sentry service, responsible for issuing certificates from a shared Certificate Authority (CA) or "Trust Anchor."

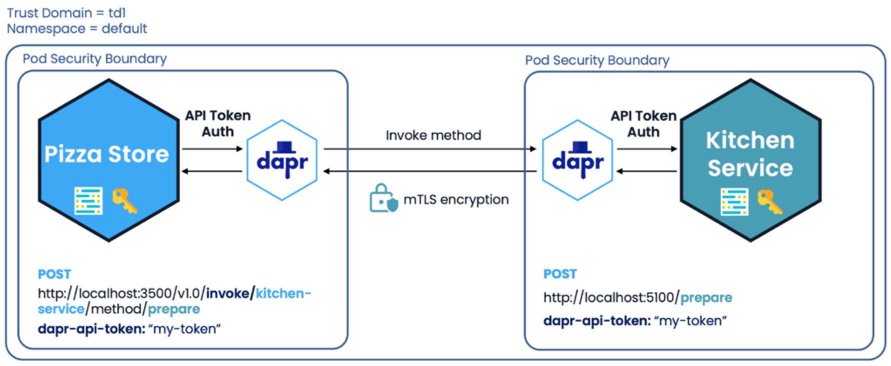
The public CA is shared among all Dapr applications so they can verify peer identities. Sentry cryptographically attests application identities by validating the client's submitted Kubernetes bound service account token and cross-referencing the requesting Pod's App ID annotation. Dapr's workload identity root of trust, or "bottom turtle," is the Kubernetes platform, and the pod can be considered the security boundary for the application.

Zero Trust App Invocation

Once App IDs are created, policies for cross-application invocation can be applied. These policies can vary in detail, from a simple access-control list containing the apps allowed to invoke the callee app, to specifying the operation, HTTP verb and path that a callee app can invoke. When policies are being checked by the Dapr Sentry service, the trust domain, namespace and App ID values of the calling application are extracted from the SPIFFE id in the TLS certificate of the calling app.

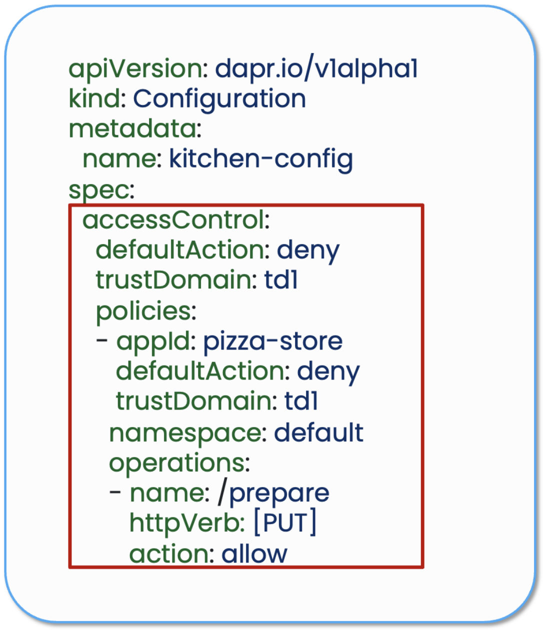
These values are then matched against the trust domain, namespace and App ID values specified in the policy specification of the callee app. If all three of these match, then the calling application is allowed to invoke the callee application at the route and operation specified. Dapr also has the concept of trust domains that create logical groupings of trust relationships and allow trust groups that are at a lower level than namespaces.

Let's look at a concrete example using the Pizza Store system. In this case, the Pizza Store service needs to directly invoke the Kitchen Service on the */prepare* method to tell the kitchen to prepare a pizza. Each of these applications is automatically given an App ID from Dapr and the communication is encrypted by default using mTLS.



Pizza Store service invoking Kitchen Service following zero trust principles

By default, though, any application is allowed to invoke any other application which does not abide by zero trust principles. To mitigate this, you need to add a Dapr configuration access control policy that enforces principle-least privilege access, thus only allowing the Pizza Store service to invoke the Kitchen Service on the */prepare* method. Service invocation policies are always applied to the callee app and are loaded at runtime into the associated Dapr sidecar.



```
apiVersion: dapr.io/v1alpha1
kind: Configuration
metadata:
  name: kitchen-config
spec:
  accessControl:
    defaultAction: deny
    trustDomain: td1
    policies:
    - appId: pizza-store
       defaultAction: deny
       trustDomain: td1
      namespace: default
      operations:
      - name: /prepare
         httpVerb: [PUT]
         action: allow
```

Dapr configuration access policy for Kitchen Service

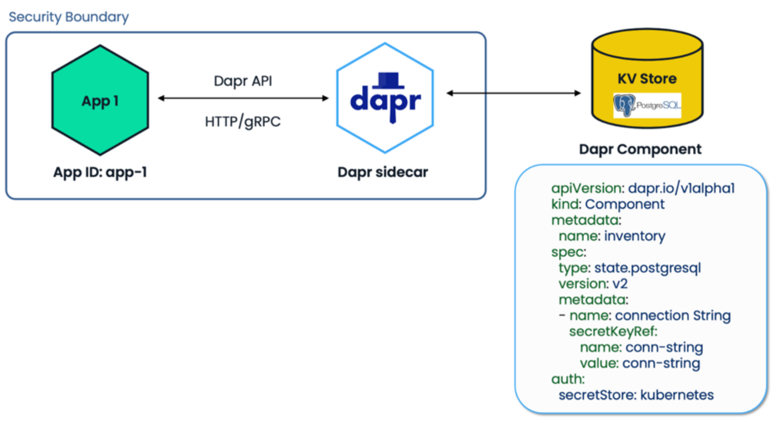
This Dapr configuration access control policy ensures that no other applications can call the Kitchen Service app, and even the Pizza Store service can only call the Kitchen Service app on the */prepare* method using the PUT HTTP verb.

There are additional security features that can be added as well, including the ability to enforce API Token Authentication on the Dapr APIs. This is useful when the Dapr APIs are exposed outside of a system through a proxy or used by frontend applications. To lock down the Dapr APIs even further, you can selectively enable the Dapr APIs that can be called by a Dapr sidecar. This ensures that the application and sidecar can only access the minimal set of infrastructure resources needed and reduces the attack surface further.

Dapr Component Secure Infrastructure Access

Another important concern in zero trust security for distributed systems is ensuring that the underlying infrastructure resources are accessed securely from the code. Of course, there are many connection authentication and encryption best practices here but even with these in place, codebases still end up with credentials scattered in source code or stored as environment variables. This can lead to a greater surface area of attack and an increased burden placed on developers to manage these.

This is where the Dapr Component model comes in. Dapr Components are described with a YAML interface for infrastructure resources and used by the Dapr APIs to perform various operations on the underlying infrastructure resources. The Component model removes the infrastructure dependency from the source code by containing the connection configuration details needed to access the infrastructure in the Component file, which is loaded by Dapr dynamically at runtime. There are over 120 component specifications, all open source, contributed by the community.



```
apiVersion: dapr.io/v1alpha1
kind: Component
metadata:
  name: inventory
spec:
  type: state.postgresql
  version: v2
  metadata:
  - name: connection String
    secretKeyRef:
      name: conn-string
      value: conn-string
auth:
  secretStore: kubernetes
```

Dapr Component model for infrastructure access

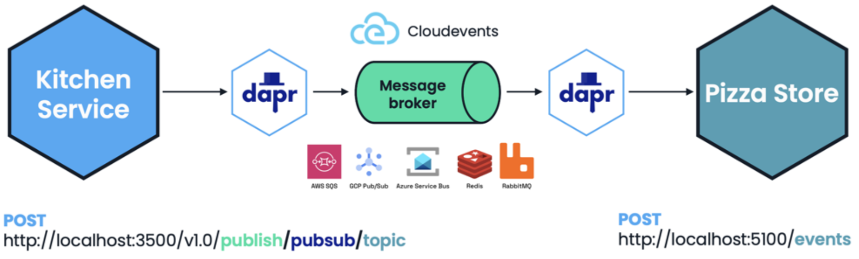
Dapr Components can be used to restrict access to infrastructure resources at runtime. This is called Component scoping in Dapr and allows:

- Principle least privilege access to infrastructure configured in the app.
- Infrastructure access control lists on top of network restriction.
- Namespace and trust domain boundary enforcement.

Zero Trust Async Communication

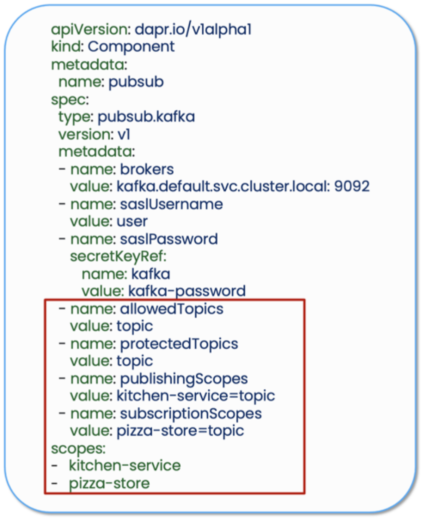
The combination of Dapr App IDs described previously and scoping Dapr components allows only required applications to access critical underlying infrastructure resources such as a production database or message broker. This is critical when adhering to zero trust principles, as it reduces the surface area of attack for bad actors when attempting to access valuable data. In the case of asynchronous communication via a message broker, only the applications publishing on the broker and subscribing to it should be allowed access.

Let's look at an example in the Pizza Store system. The kitchen service needs to communicate with the Pizza Store service to let the customer know that the pizza has finished cooking. For this purpose, the system uses asynchronous communication via a message broker. The Kitchen Service takes advantage of the Dapr Publish and Subscribe API and a Dapr *pubsub* component specification to publish a message on a topic on the message broker, achieving at least one delivery semantics and removing all dependencies on the message broker from the source code. The Pizza Store service subscribes to the topic and receives messages using the */events* method to notify the user.



Asynchronous communication using Dapr Publish and Subscribe API

By default, any application can publish or subscribe to this message broker, so we need to restrict this to adhere to zero trust principles. There are several ways to limit which applications can publish and subscribe to the message broker infrastructure, starting with scoping. Adding the App IDs of the Kitchen Service and Pizza Store services to the Dapr Component file ensures that only the sidecars for these apps will load this Component at runtime — giving access to only those applications. Dapr publish and subscribe access policies can get even more granular by specifying the topics allowed on the message broker and lists of applications that can perform publish and subscribe operations. This ensures that a bad actor within the system will not be granted access to the underlying infrastructure resource to create new topics, publish messages or receive data.



```yaml
apiVersion: dapr.io/v1alpha1
kind: Component
metadata:
  name: pubsub
spec:
  type: pubsub.kafka
  version: v1
  metadata:
  - name: brokers
    value: kafka.default.svc.cluster.local: 9092
  - name: saslUsername
    value: user
  - name: saslPassword
    secretKeyRef:
      name: kafka
      value: kafka-password
  - name: allowedTopics
    value: topic
  - name: protectedTopics
    value: topic
  - name: publishingScopes
    value: kitchen-service=topic
  - name: subscriptionScopes
    value: pizza-store=topic
scopes:
-  kitchen-service
-  pizza-store
```

Kafka message broker Dapr Component with access control for Kitchen and Pizza Store services

In this example, using a Kafka component description, it specifies that only the Kitchen Service can publish a message on the *topic* and only the Pizza Store service can subscribe to it — meaning that no other apps, even within the system, can connect to the Kafka message broker.

Conclusion

In summary, Dapr improves the zero trust security posture of distributed systems out of the box by assigning application identities to all apps, ensuring that mTLS is enabled by default for all interservice and infrastructure communication. Several additional security features can be layered on top, including adding Dapr Configuration policies for restricting service invocation between apps and configuring scopes for Dapr Components. Read more about how Dapr was designed with security in mind here.

Watch this session recorded at App DeveloperCon, Sleep Better at Night: Dapr's Approach to Secure Cloud Native Development, to see zero trust security in action with Dapr, complete with live demo scenarios.

XENVECTOR

https://www.xenvector.com
Austin Texas
email : webreach@xenvector.com

XenVector Specializes in Engineering lead project delivery for Platform Engineering (AKS/Azure App Service), Generative AI (OPEN-AI / Azure AI) and Azure Security projects (Entra ID, Defender XDR, Sentinel SIEM) aimed at helping HealthCare and Manufacturing SME companies rapidly solve business problems using proven solution accelerators.