AWS
re:Invent

GPSTEC413

Best practices and design patterns for building IoT solutions on AWS

Craig Williams
Principal Solutions Architect, IoT
Amazon Web Services

Priyanka Jindal
Senior PMT, IoT
Amazon Web Services

AWS re:Invent

aws

What to expect with this session?

Session agenda

- Take a look at patterns that emerge in IoT solutions
- Scaling your solution, how are those downstream services?
- Critical message and event processing
- Device security best practices
- AWS IoT Atlas

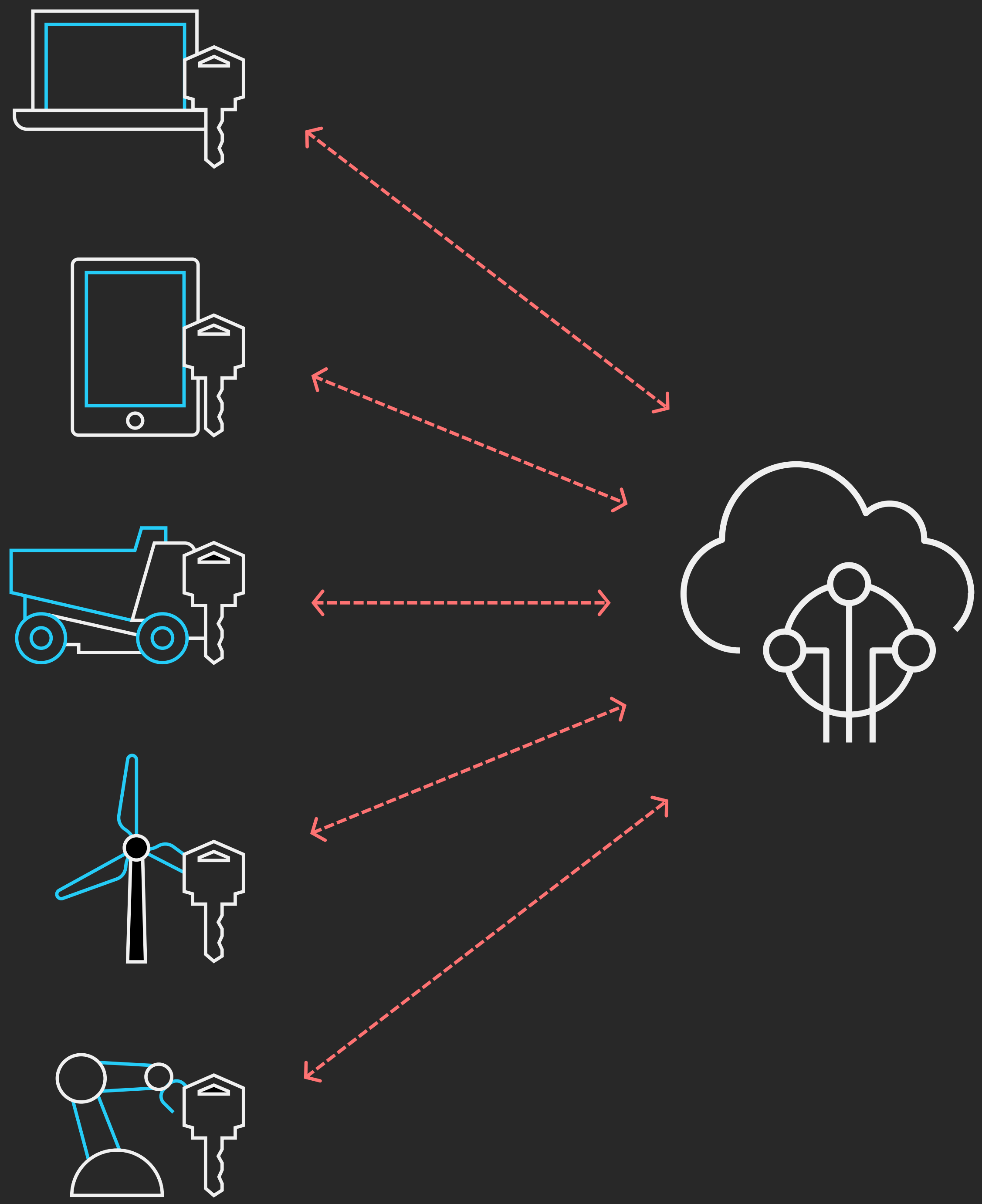
Scaling your IoT solution

Scaling

Understanding your scaling architecture and throttle points

- Endpoint scaling via a platform architecture
- Downstream service considerations
- Think about message recovery
- Batching data for efficiency and cost optimization
- Does you data belong on a broker?

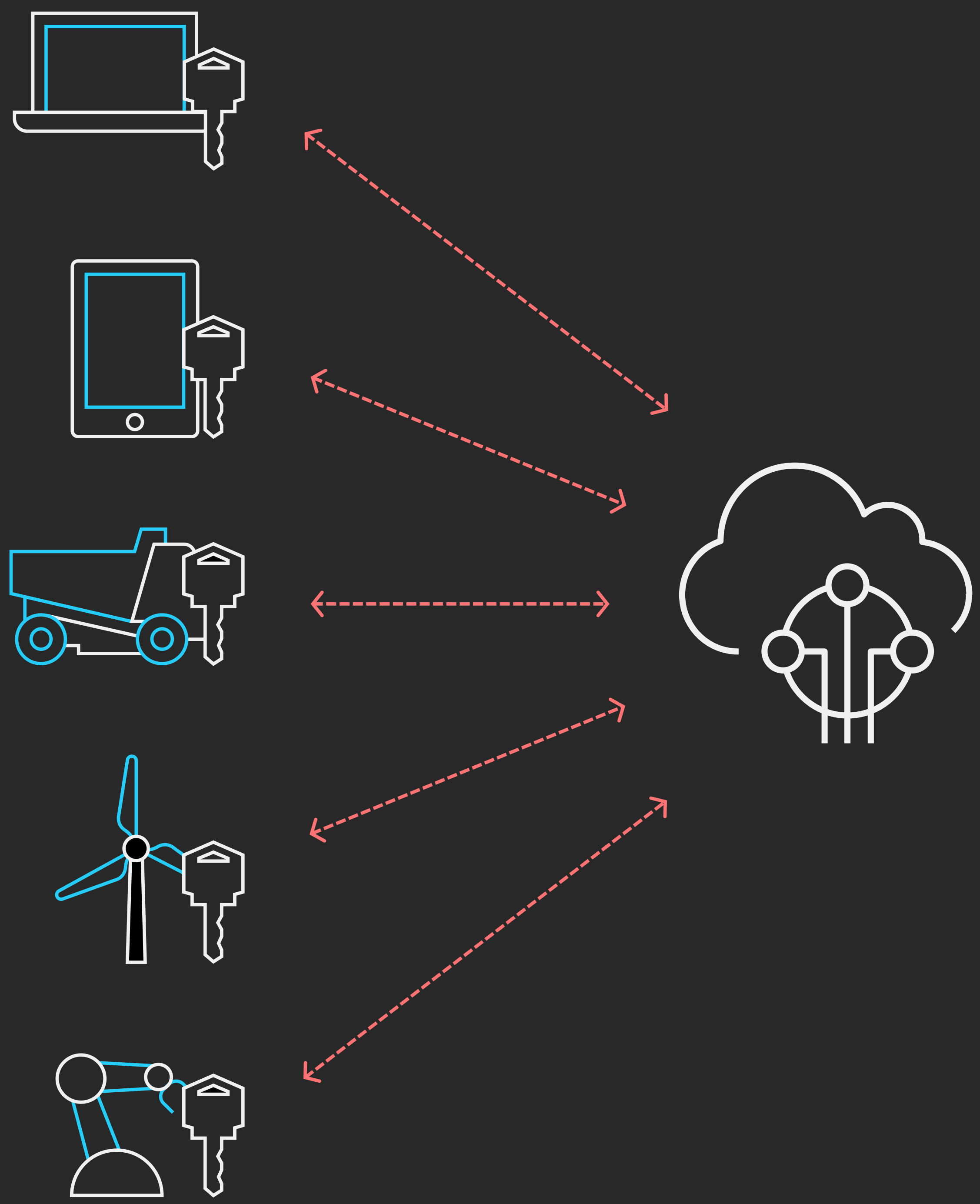
Note: AWS IoT services are all built as platforms, handling the scaling of millions of devices and events



Patterns for recovery

Understand message failure points and recovery options

- Buffering messages
- Queuing messages
- Known the difference
- Implementation options

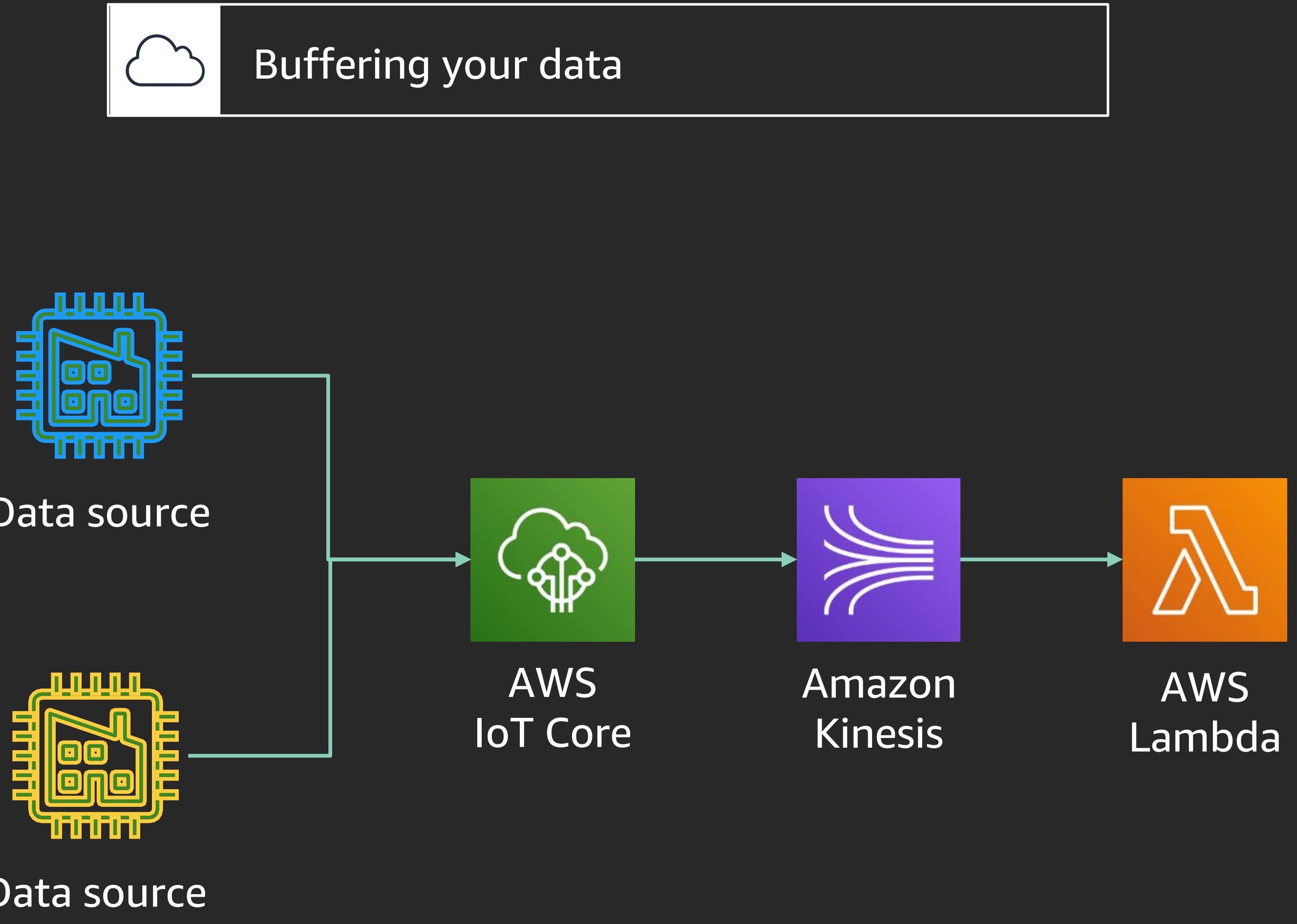


Buffering data

Have a safe way to hold onto data until you can process it

- Handle data spikes
- Allow downstream services to scale
- Hold data if you’re throttled
- Batch and cost optimization

Note: Once data is removed from a stream, it cannot be recovered. If it needs to be, consider using a queue.



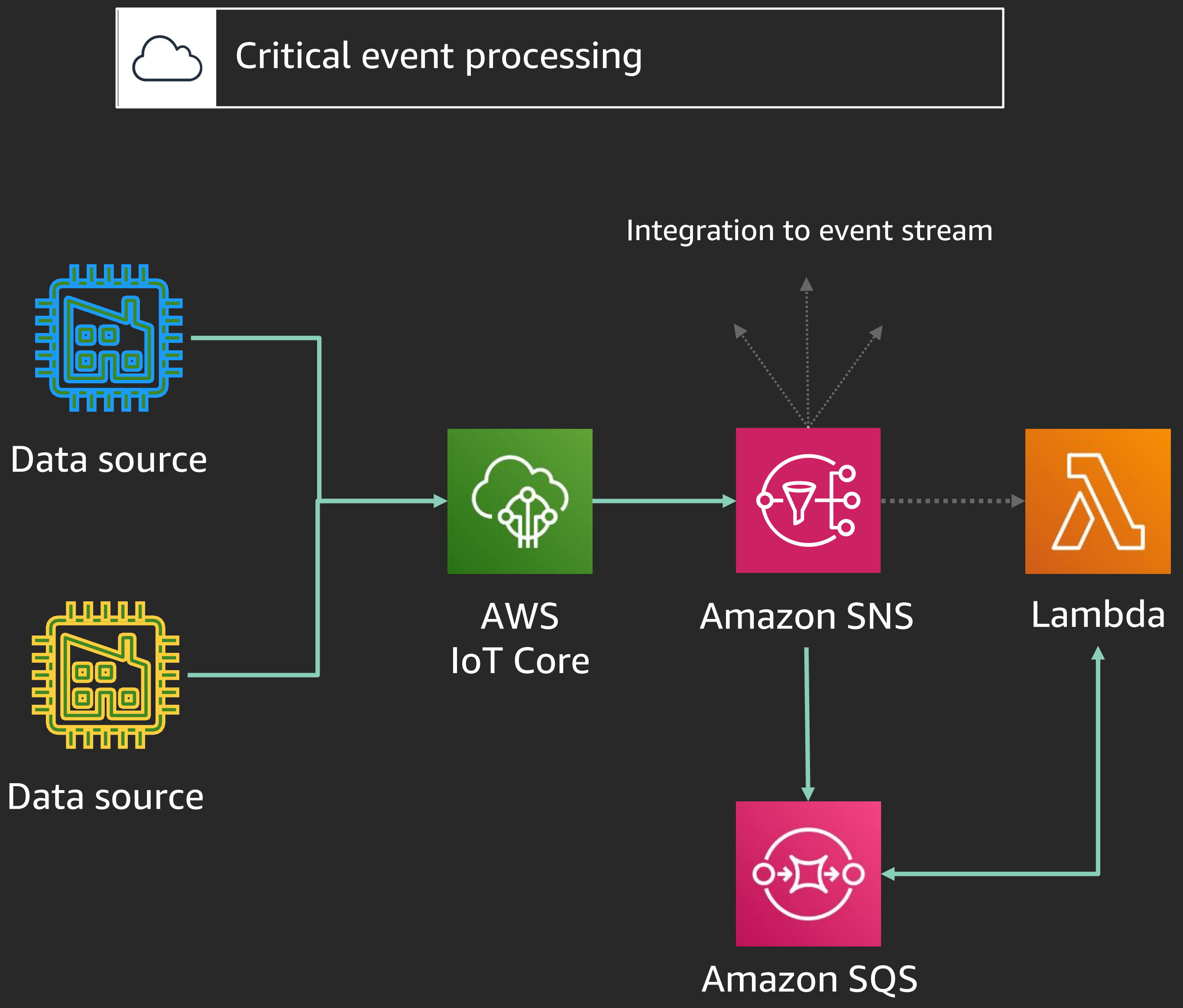
Critical event processing

Queue data and fan out

Have a safe way to hold onto data until you can process it

- Handle data spikes
- Allow down stream services to scale
- Hold data if you're throttled
- Batch and cost optimization

Note: Having a mechanism to fan out helps prevent a fragile downstream architecture



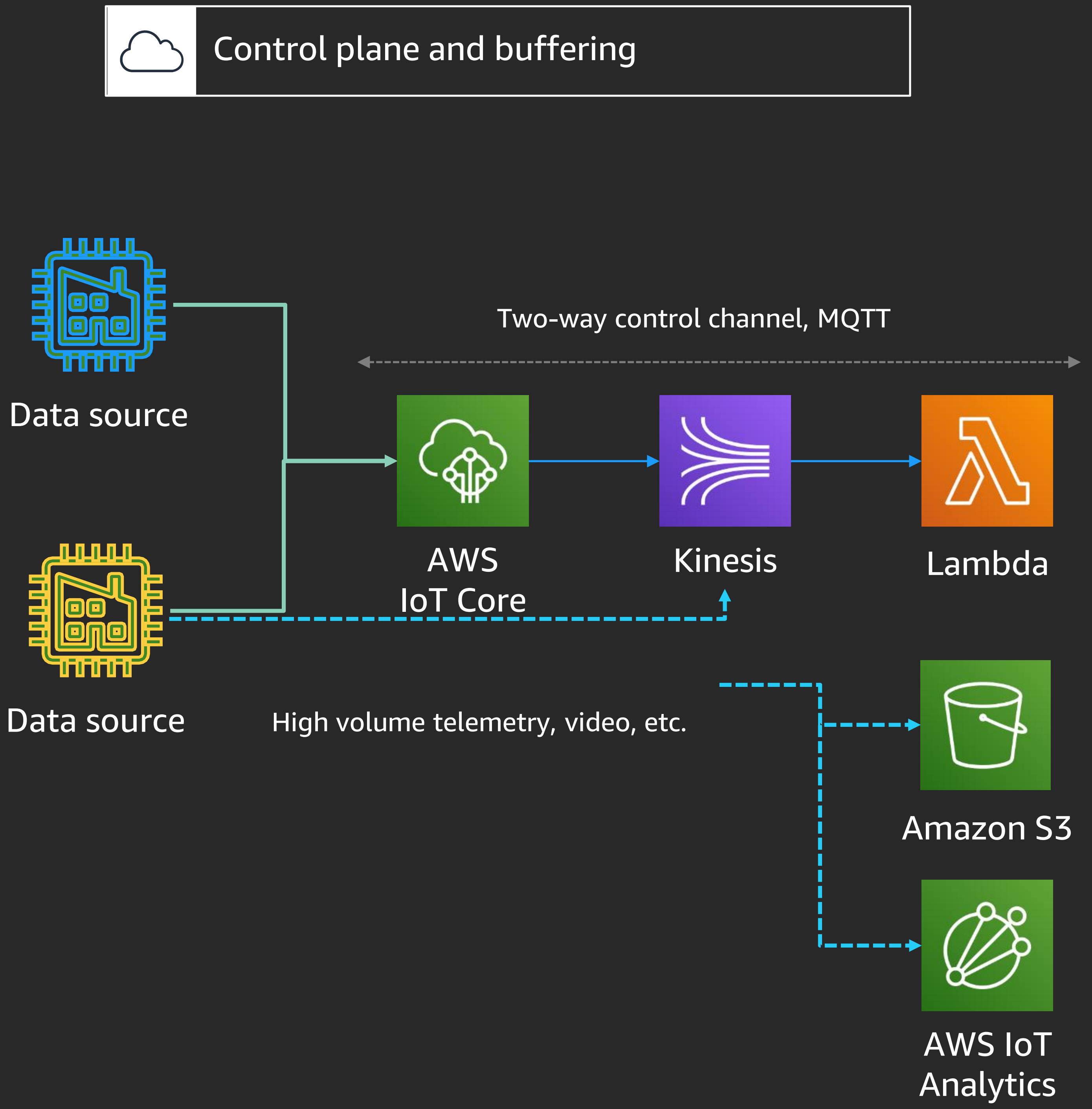
General designs

Control plane

Using a control plane to set up high-volume telemetry streaming

- Not all data belongs on a broker
- Control is over the broker
- Telemetry goes directly to your buffering service
- Can be more cost effective
- Avoids potential limits

Note: The downstream compute component can process from the broker or a stream

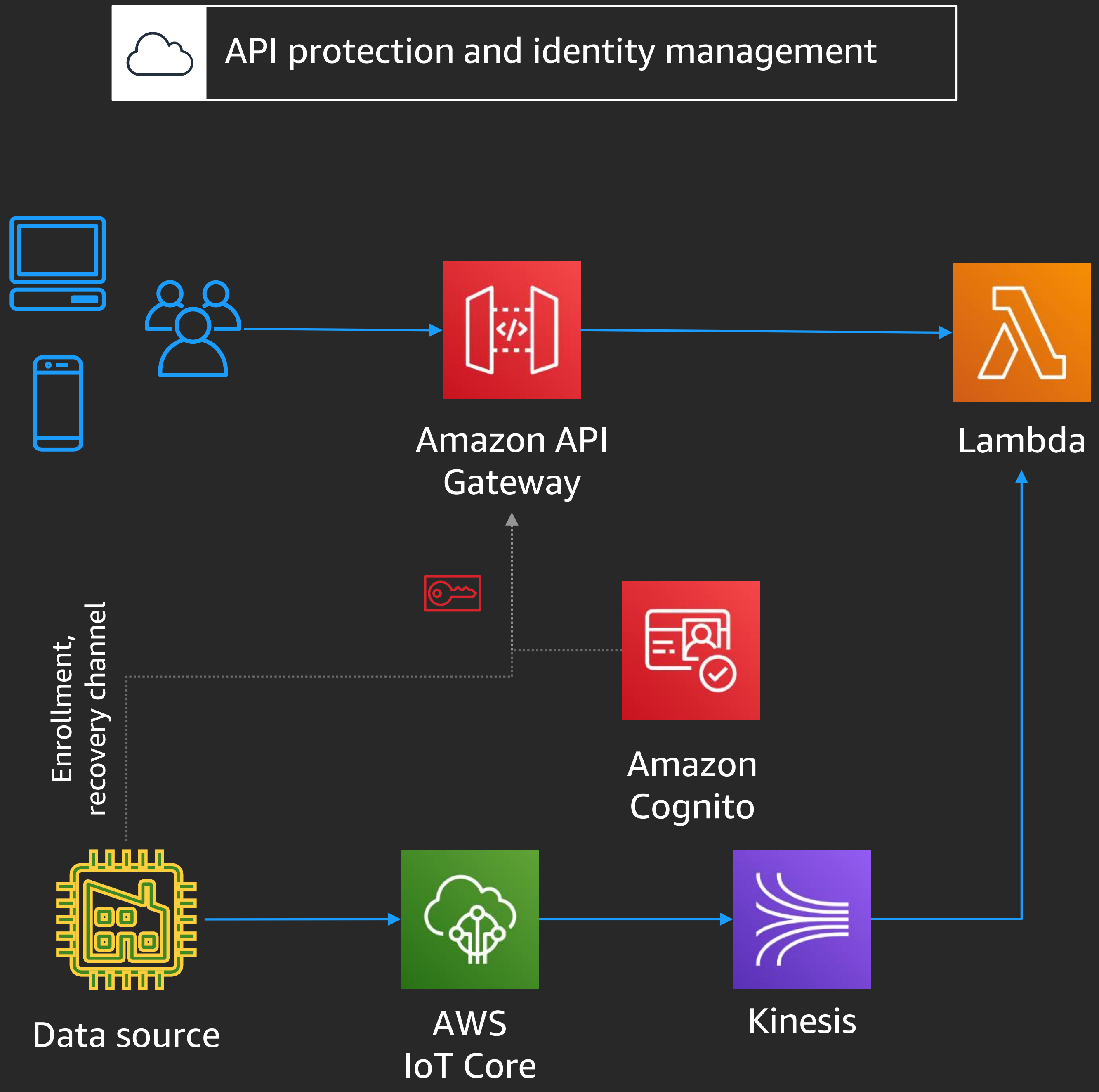

Control plane and buffering
Data source
Data source
Two-way control channel, MQTT
AWS IoT Core
Kinesis
Lambda
High volume telemetry, video, etc.
Amazon S3
AWS IoT Analytics

Protect your APIs

Never expose backend APIs directly and use authentication services

- Unified entry point for control systems and applications
- Protect downstream APIs with TPS throttling and DDOS protection
- Use platform identity services to authenticate API requests

Note: The more AWS services you can put in front of your services and/or APIs, the better protection you will have

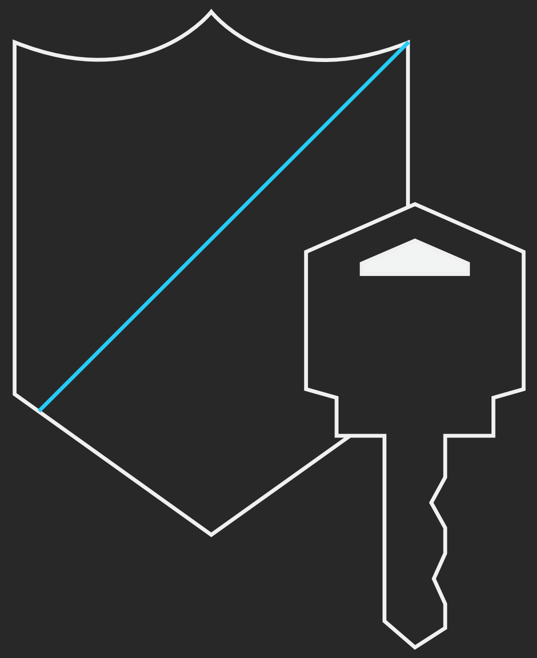


When should you rotate device certificates?

Risks of certificate rotation

Don’t risk exposure with unnecessary rotation

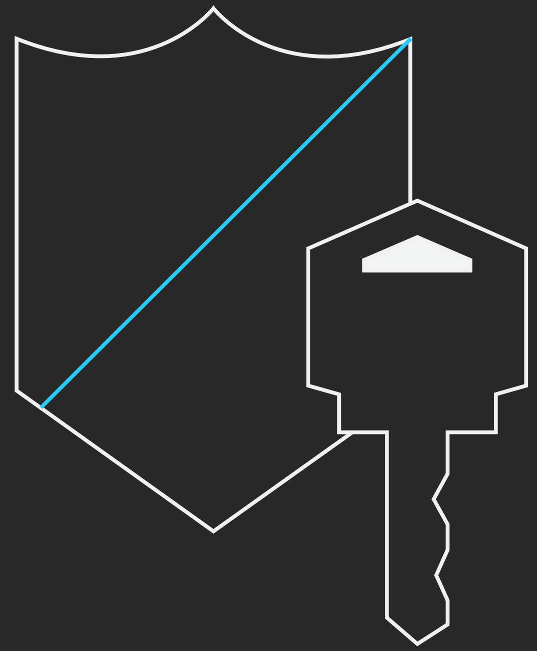
- You could brick the device and then require physical access to recover it
- Anytime you move certificates and keys, you risk exposure
- You aren’t guaranteeing device security!



Security best practices

Know what's happening with your devices

- Know what's going on with your device vs. rotate and pray
- Do not rotate certificates on an arbitrary interval
- Have mechanisms for detecting:
  - Multiple connections using the same certificate
  - Anomalies with device data
  - Anomalies with device hardware, unexpected active interfaces such as SPI/I2C or even open ports
- Faulty-device detection—it's not always a security issue

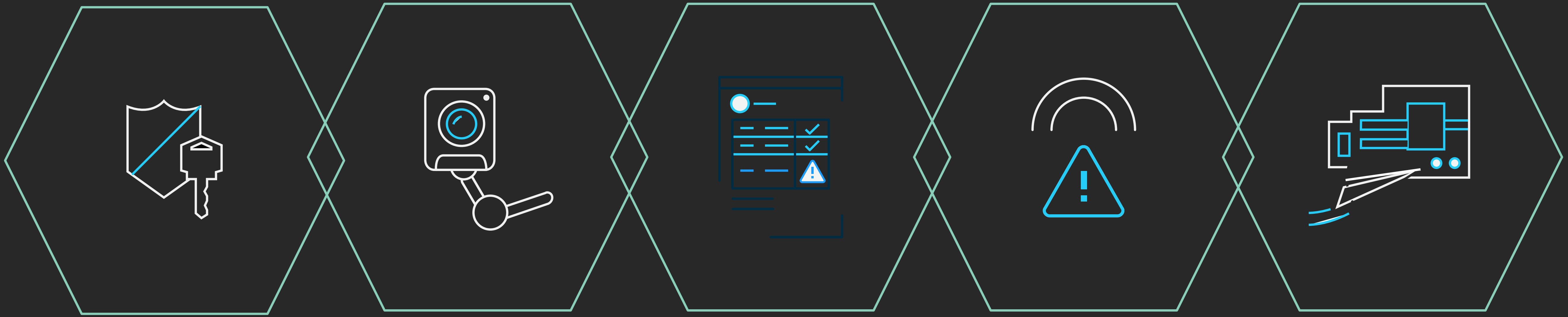


Use hardware security modules/TPMs, PKCS#11

How do I ensure that my connected devices stay secure?

AWS re:Invent

AWS IoT Device Defender



Audit

Validate that IoT configuration is secure

Security dashboard

Continuously monitor configurations to understand security posture

Detect

Detect anomalies in device behavior

Alerts

Know when and what to investigate

Mitigate

Remediate potential issues

AWS IoT Atlas

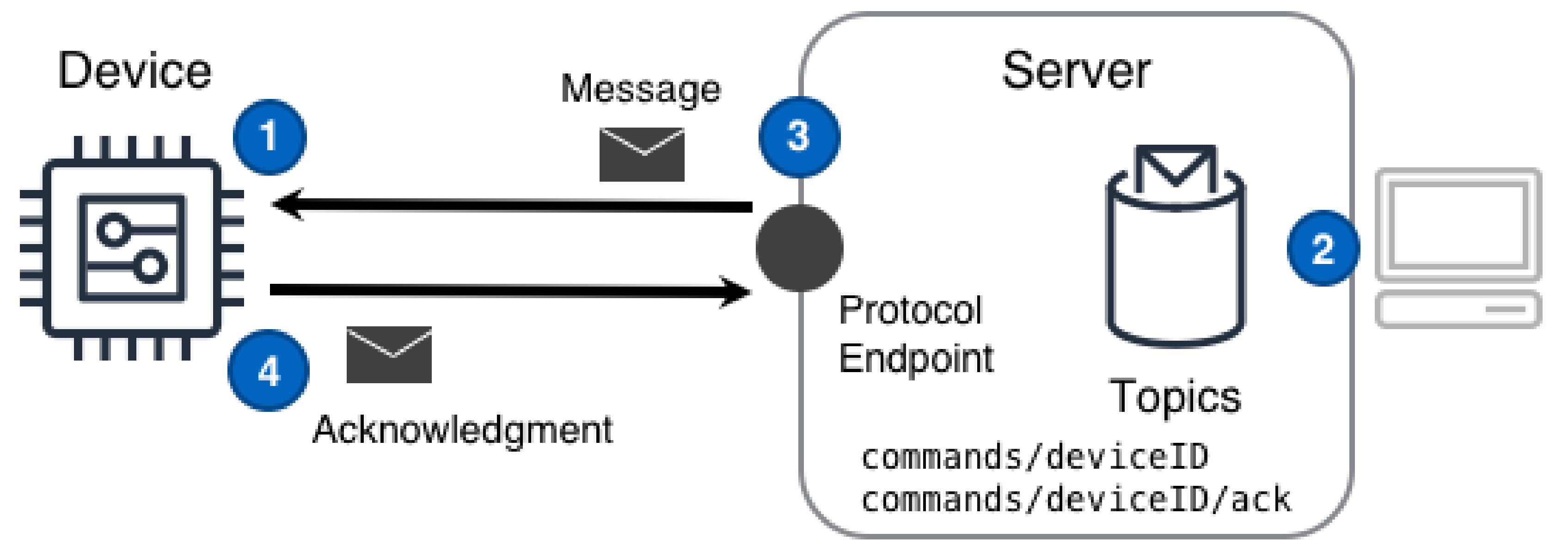
AWS IoT Atlas

A collection of cloud-agnostic best practices and patterns

https://iotatlas.net

Command

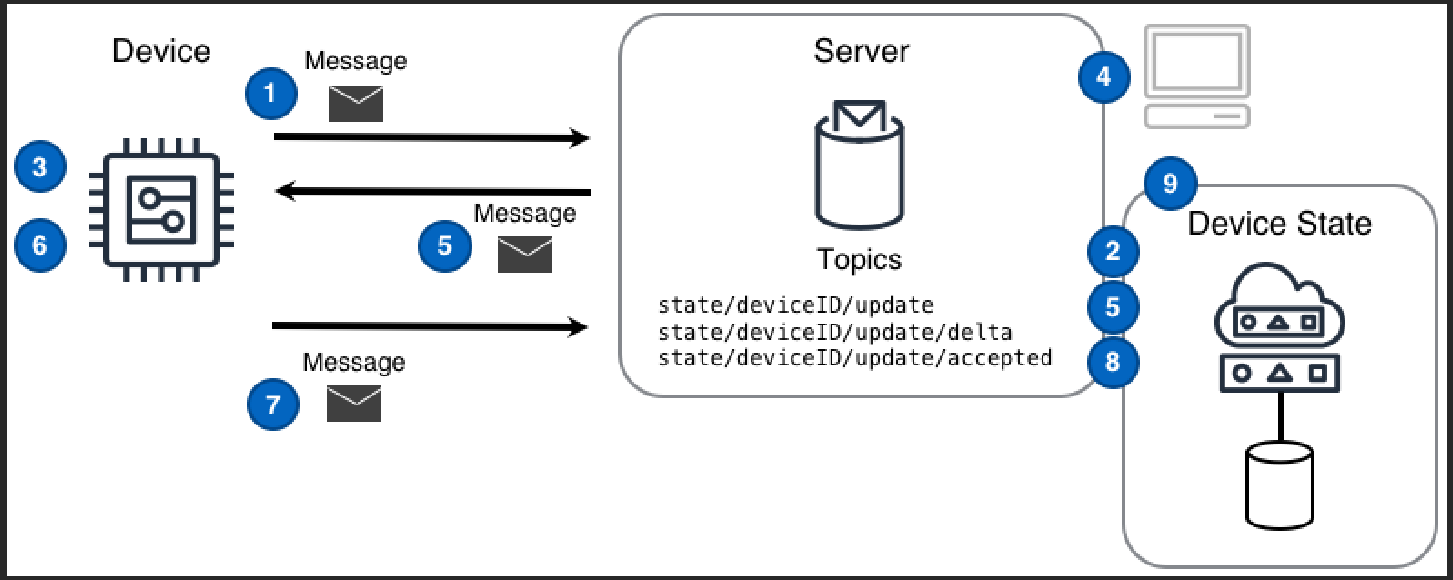
A requesting entity reliably asks a device to perform a single action, with acknowledgement of status



1. A device configures itself to communicate with a protocol endpoint so that Command messages can be sent and received
2. A component of the solution publishes a Command message targeted at one or more devices
3. The server uses the protocol endpoint to send the Command message to each previously configured device
4. Upon completion of the Command's requested action, the device publishes a command completion message to the server via the protocol endpoint

Device state replica

A logical representation of a physical device’s reported state, or desired future state

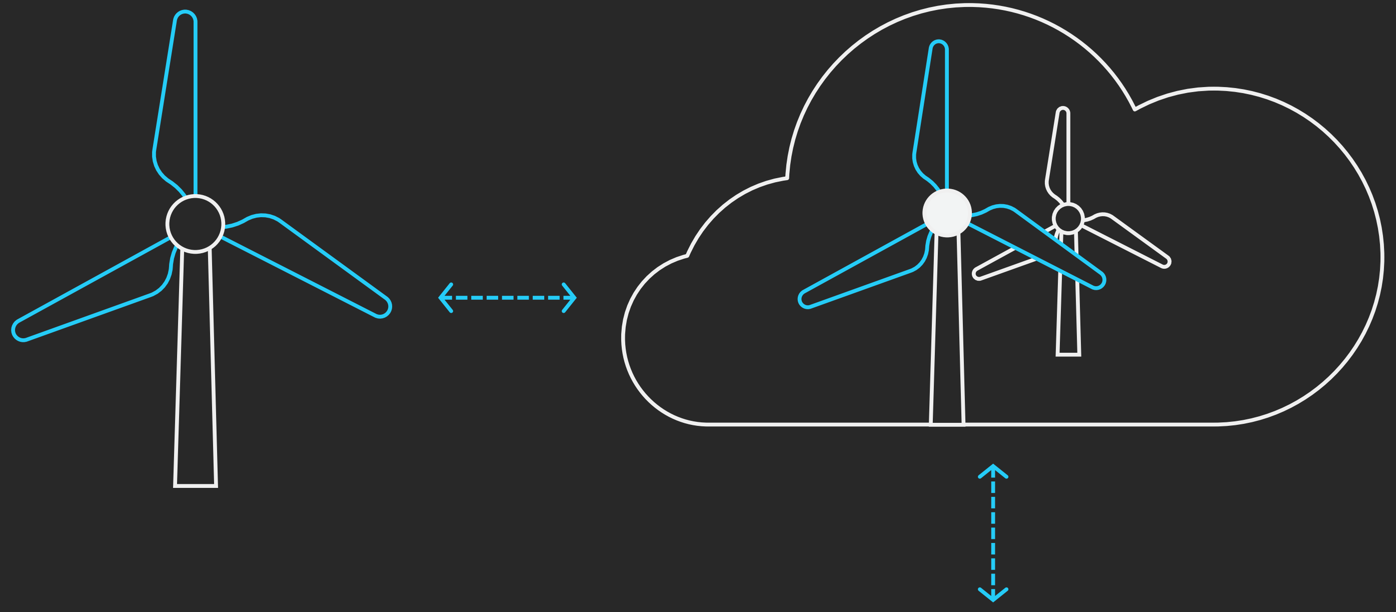


1. A device reports **initial device state** by publishing that state as a message to the state/deviceID/update topic
2. The device-state replica reads the message from the state/deviceID/update topic and records the device state in a persistent data store
3. A device subscribes to the delta messaging topic state/deviceID/update/delta upon which device-related state change messages will arrive
4. A component of the solution publishes a desired state message to the topic state/deviceID/update and the device-state replica tracking this device records the desired device state in a persistent data store
5. The device-state replica publishes a delta message to the topic state/deviceID/update/delta and the server sends the message to the device
6. A device receives the delta message and performs the desired state changes
7. A device publishes a message reflecting the new state to the update topic state/deviceID/update and the device-state replica tracking this device records the new state in a persistent data store
8. The device-state replica publishes a message to the state/deviceID/update/accepted topic
9. A component of the solution can now request the updated and current state form the device-state replica

State replica example

The DSR is the most critical feature to understand …

```
"state" : {
        “desired" : {
            "lights": { "color": "RED" },
            "engine" : "ON"
        },
        "reported" : {
            "lights" : { "color": "GREEN"  },
          "engine" : "ON"
        },
        "delta" : {
            "lights" : { "color": "RED"  }
         } },
"version" : 10
}
```



APIs

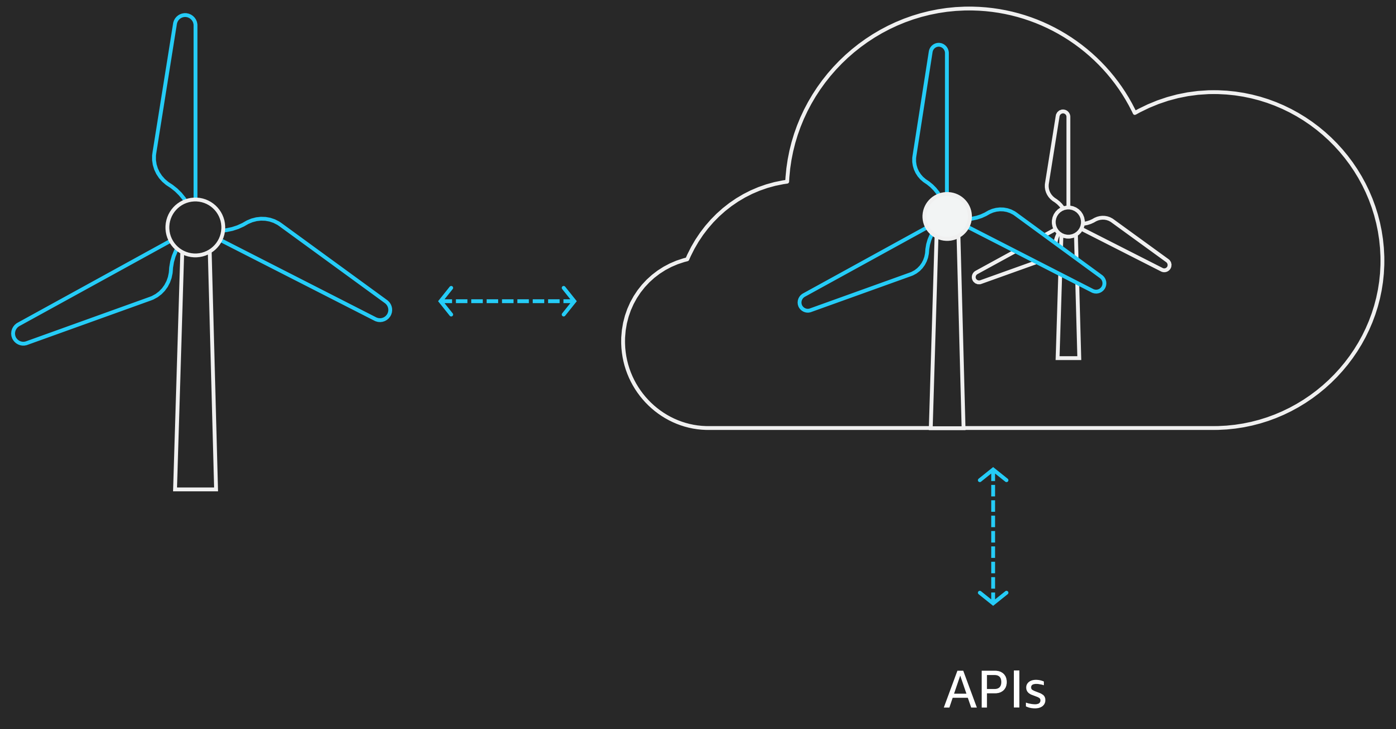
Note: AWS IoT implements the DSR as a device shadow, which is managed by AWS IoT in the cloud

DSR patterns

These are the issues the DSR is helping to avoid

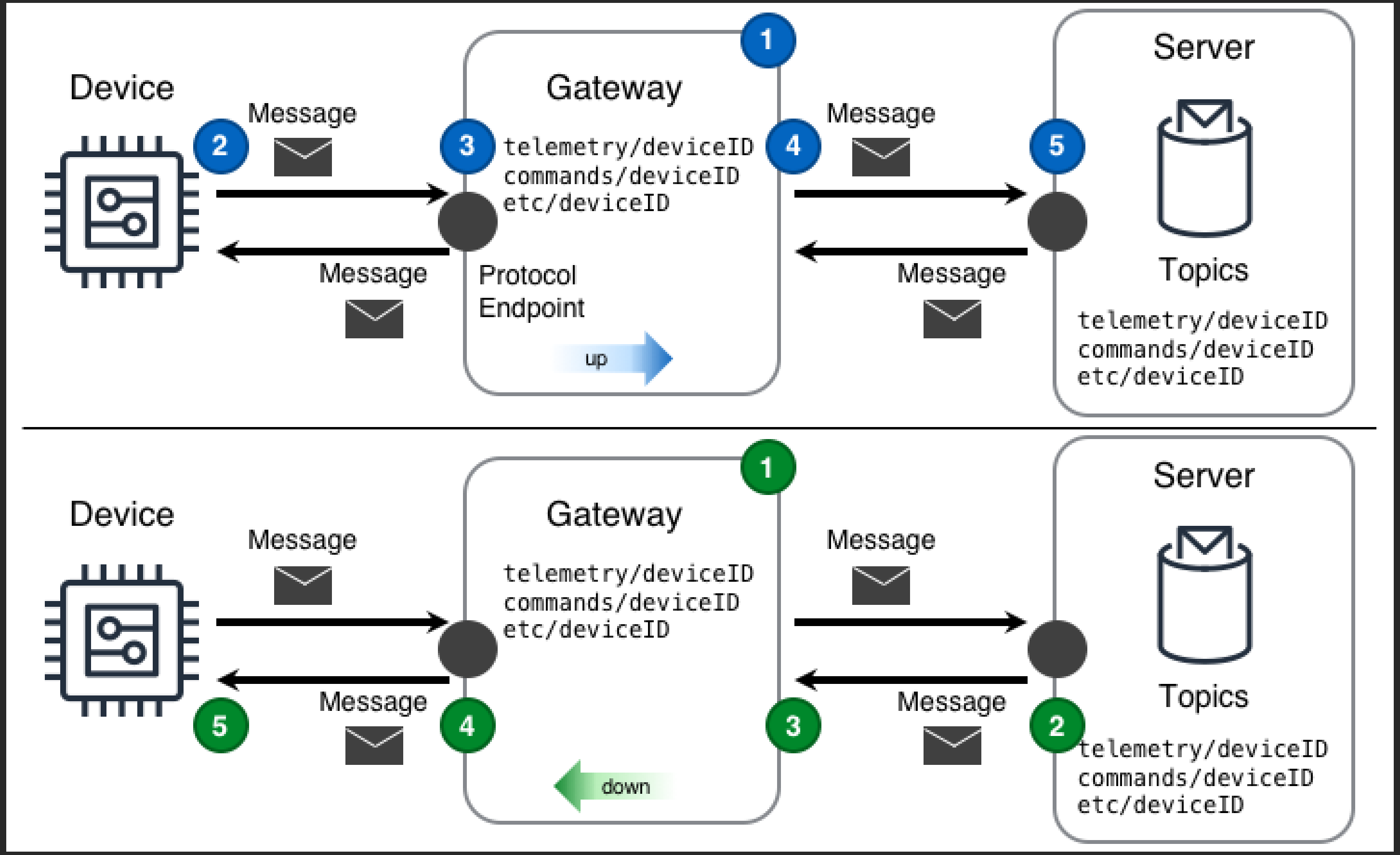
- Direct publishing – Typically developers want to do this, but beware of the pitfalls
- Devices are offline – Devices are never always online; be careful of only testing lab environments
- Messages can be unordered – At high volume, your IoT messages may arrive unordered

Note: Remember, using the shadow for command and control means we have solved the issues of message timing and reliable delivery for you, for consistent state changes



Gateway

A device acts as an intermediary between local devices as well as between devices and the cloud



IoT solutions use the gateway design to overcome the common constraints experienced by endpoints. In doing so, a gateway enables reliable and secure communication with otherwise inaccessible endpoints. Additionally, a gateway enables continued isolation between the local endpoints and cloud connectivity.

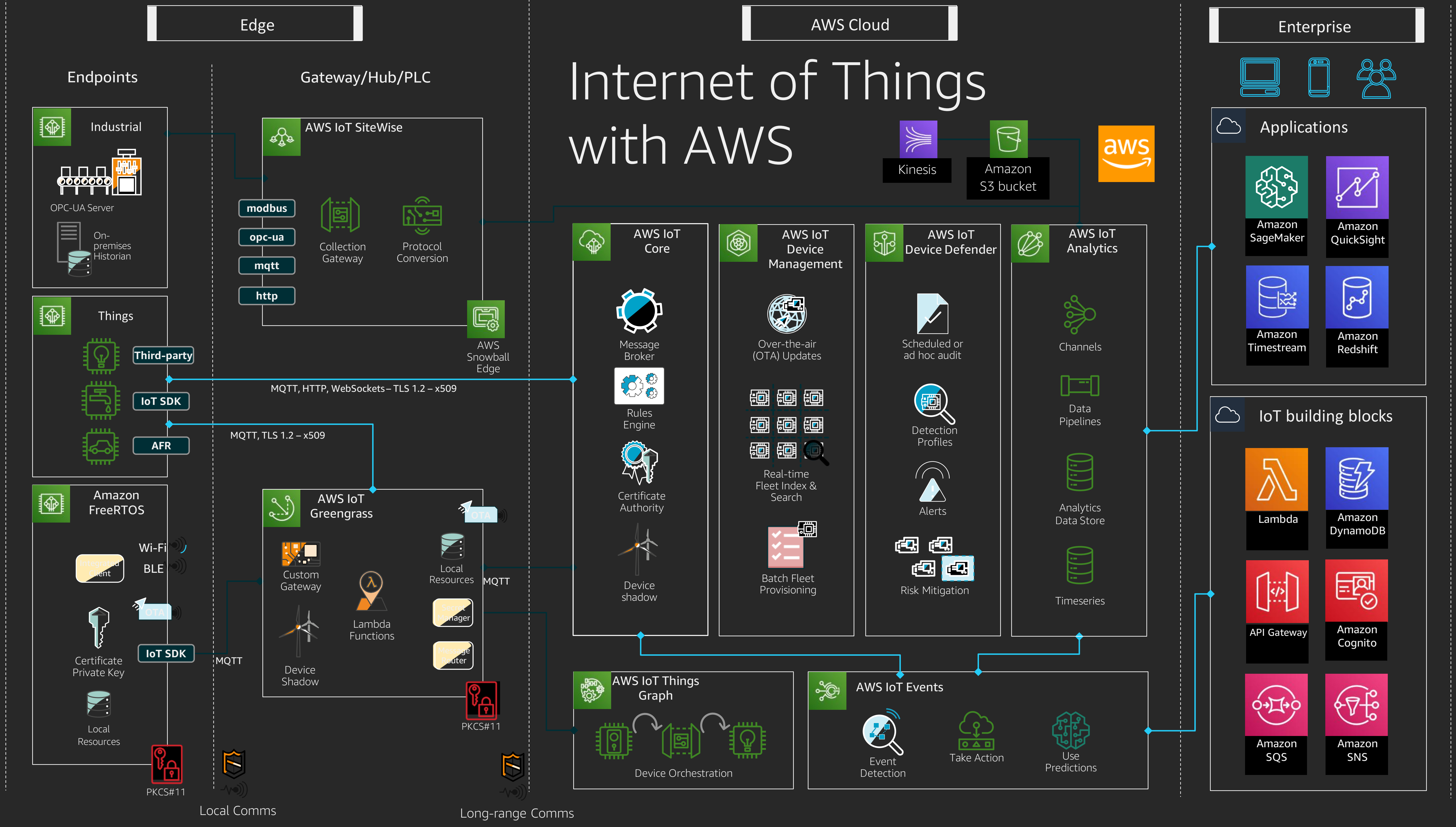
IoT Atlas – Contributions welcome

Contributions welcome

We welcome community contributions via our public GitHub repository and pull requests

IoT on AWS

Internet of Things with AWS



Edge

Endpoints

Industrial
- OPC-UA Server
- On-premises Historian

Things
- Third-party
- IoT SDK
- AFR

Amazon FreeRTOS
- Integrated Client
- Wi-Fi
- BLE
- OTA
- Certificate Private Key
- IoT SDK
- Local Resources
- PKCS#11

Gateway/Hub/PLC

AWS IoT SiteWise
- modbus
- opc-ua
- mqtt
- http
- Collection Gateway
- Protocol Conversion
- AWS Snowball Edge

AWS IoT Greengrass
- OTA
- Custom Gateway
- Local Resources
- Lambda Functions
- Secret Manager
- Device Shadow
- Message Router
- PKCS#11

MQTT, HTTP, WebSockets – TLS 1.2 – x509

MQTT, TLS 1.2 – x509

MQTT

MQTT

Local Comms

Long-range Comms

AWS Cloud

Kinesis

Amazon S3 bucket

aws

AWS IoT Core
- Message Broker
- Rules Engine
- Certificate Authority
- Device shadow

AWS IoT Device Management
- Over-the-air (OTA) Updates
- Real-time Fleet Index & Search
- Batch Fleet Provisioning

AWS IoT Device Defender
- Scheduled or ad hoc audit
- Detection Profiles
- Alerts
- Risk Mitigation

AWS IoT Analytics
- Channels
- Data Pipelines
- Analytics Data Store
- Timeseries

AWS IoT Things Graph
- Device Orchestration

AWS IoT Events
- Event Detection
- Take Action
- Use Predictions

Enterprise

Applications
- Amazon SageMaker
- Amazon QuickSight
- Amazon Timestream
- Amazon Redshift

IoT building blocks
- Lambda
- Amazon DynamoDB
- API Gateway
- Amazon Cognito
- Amazon SQS
- Amazon SNS

Thank you!

Craig Williams

craiwill@amazon.com
https://www.linkedin.com/in/typemismatch/
@manicasteroid

Priyanka Jindal

prjindal@amazon.com

Please complete the session survey in the mobile app.

AWS re:Invent

aws