CompSci 401: Cloud Computing

Autopilot: Automatic Data Center Management

Prof. Ítalo Cunha

昆山杜克大学
DUKE KUNSHAN UNIVERSITY

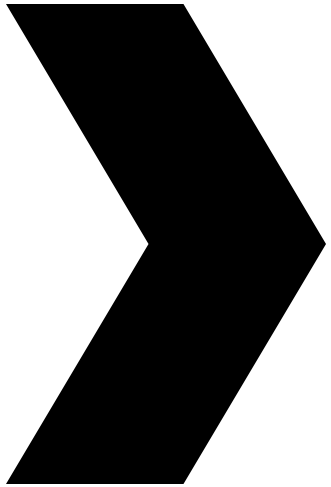
Growing infrastructure and increasing complexity

- Total number of servers is increasing
- Network is growing in number of devices and complexity
- More applications with richer interactions

Opportunity

Growth of data center capacity

Many new applications and back-end services



New infrastructure for data center management

Automatic data center management

- Intelligent software

- Replace repetitive work by operations staff
- Reduce operational expenses (OPEX)
- Avoid human errors, increase reliability

Limitations and assumptions

- Autopilot starts from a fresh application base
- Applications need to conform with Autopilot's design principles
- Automation: Basic services to keep a data center operational
  - Provisioning
  - Deployment
  - Monitoring
  - Hardware lifecycle (repair and replacement)
- No orchestration: Policy is left to individual applications
  - Scheduling
  - Placement
  - Load balancing
  - Failure detection and repair

Hardware and fault-tolerance

- Original approach: Reliable systems built on fault-tolerant hardware
  - Costly
  - Fault-tolerant hardware *will* fail → requires higher-layer resiliency mechanism
- Autopilot: Assumes commodity computers
  - Lower cost
  - Hardware is more prone to failures → use higher-layer resiliency mechanisms

Fault tolerance in Autopilot

- System must continue operating under failures
  - Fraction of computers may be down or misbehaving
- Vital state must be replicated
- Fail-over must be automatic
- Minimize critical dependencies between components
  - Avoid single points of failure and cascading effects

Fault tolerance assumptions

- Non-Byzantine failures
  - Bugs may cause machine to misbehave
    - Non-colluding
    - Affects all machines instead of a few
- Failures may happen in any combination
  - Including pathological ones
- Forced termination is the *only* exit mechanism for services
  - Enables liberal use of assertions
  - No need to recover a failing service instance
- Applications must abide by the same principles
  - Easy to restart and support for forced termination

Simplicity for reliability and maintenance

- Conservative design principles
  - Prefer simple solutions even if less efficient
  - Avoid unecessary optimization and generality
- Simplicity in the context of the entire system
  - Solution may be simple, but hard to integrate, test, and deploy
- Plain-text version-controlled configuration files
  - Keep audit trail of configuration changes
  - Discourage interactive control channels
- Correctness trumps simplicity
  - Will embrace complexity if correctness is at stake

Hardware clusters

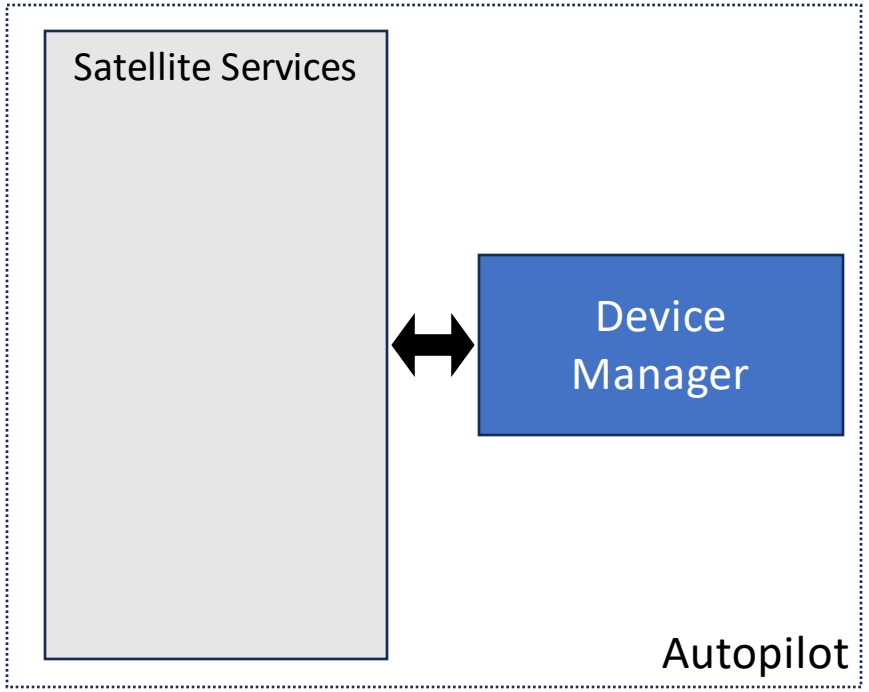
- Autopilot cluster is a set of racks
- Racks may have different server “profiles”
  - Application and storage are common classes
- Racks have a ToR switch
- Servers have a management interface

- Multiple independent clusters
  - Different failure domains

Autopilot components

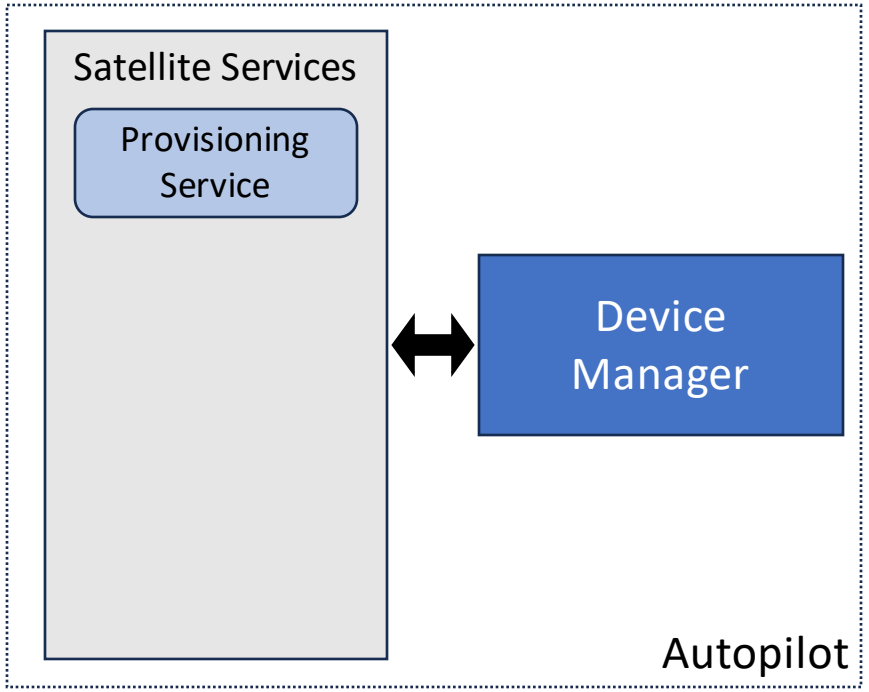
- Device manager keeps a distributed database
  - Keeps the state of the cluster, managed by other devices
  - Paxos consistency algorithm
  - Low latency requirements (tens of seconds)

Autopilot components



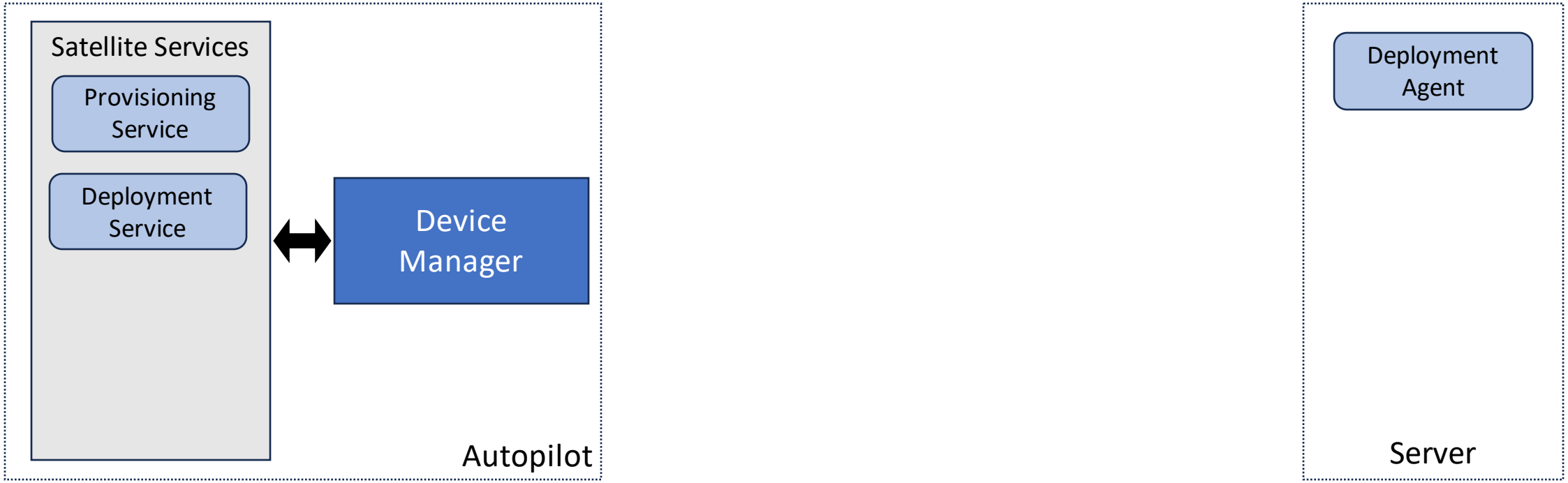
- Satellite services query device manager about state
  - Take action to steer system towards desired state
  - Can operate with weak consistency in case of failures or network partitions
  - Pull-based operation, with “kick” functionality to trigger immediate action

Autopilot components – Provisioning



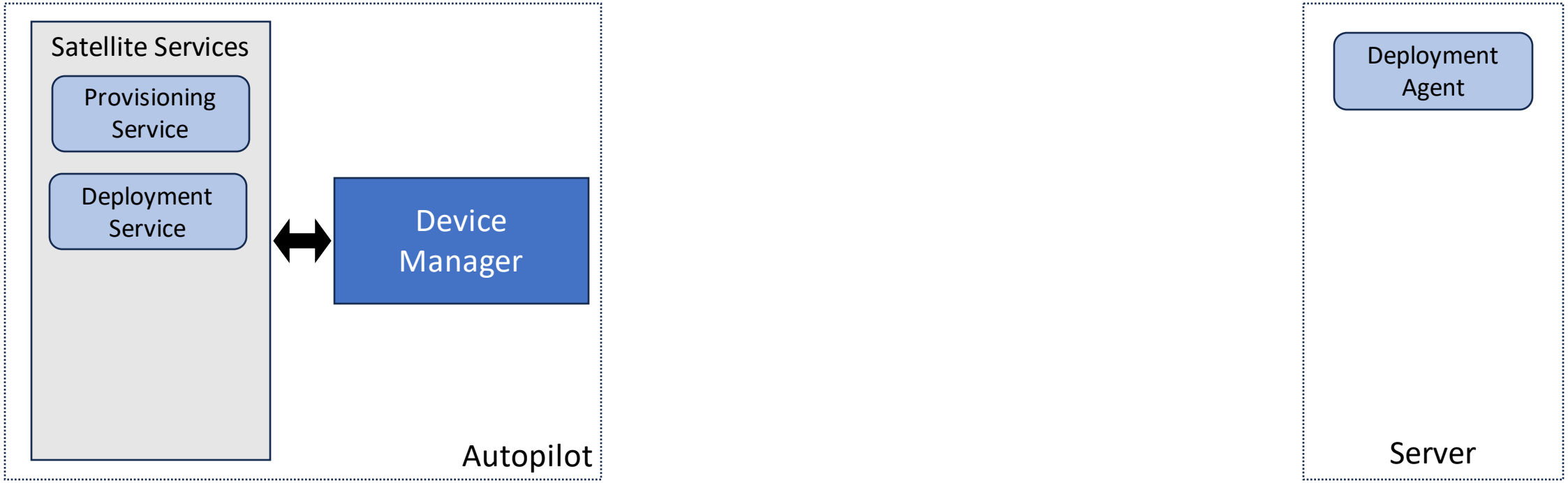
- Provisioning service boots and configures servers
  - Reads desired state from device manager, updates state after provisioning
  - Provisioned servers independently query device manager for desired state

Autopilot components – Application deployment



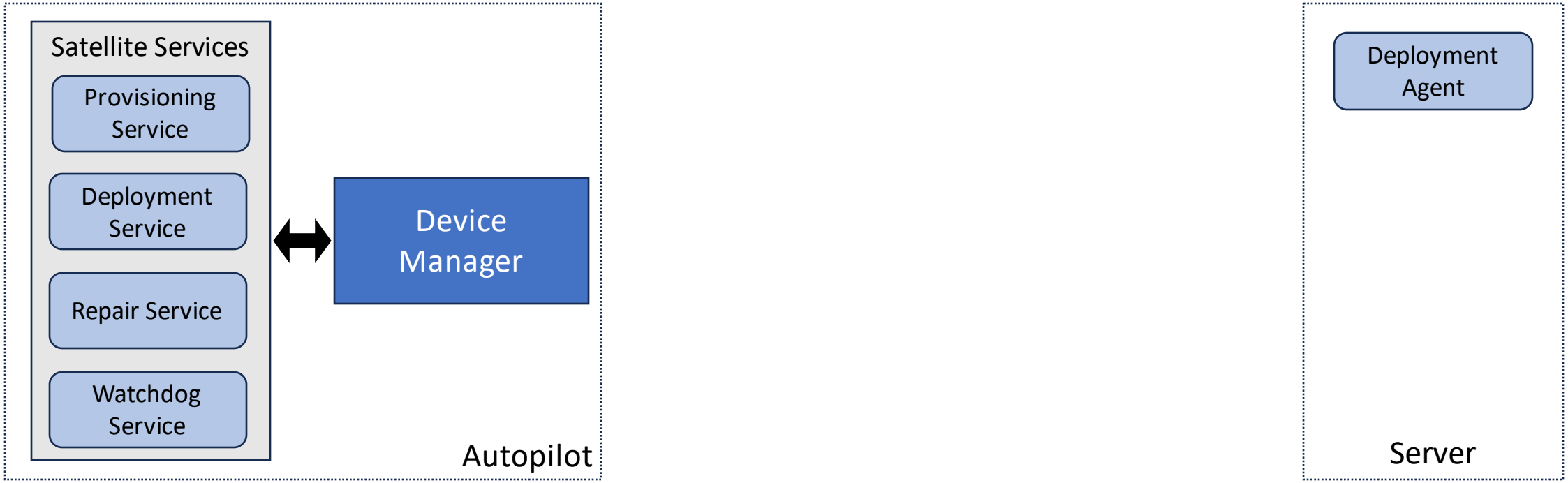
- Deployment service ensures servers converge to desired state
  - Defines manifests listing applications to store on disk and to run
  - Manifests depend on type of server (application, storage, etc)
  - Manifests read from device manager
  - Server runs agent to undertake deployment actions

Autopilot components – Application deployment



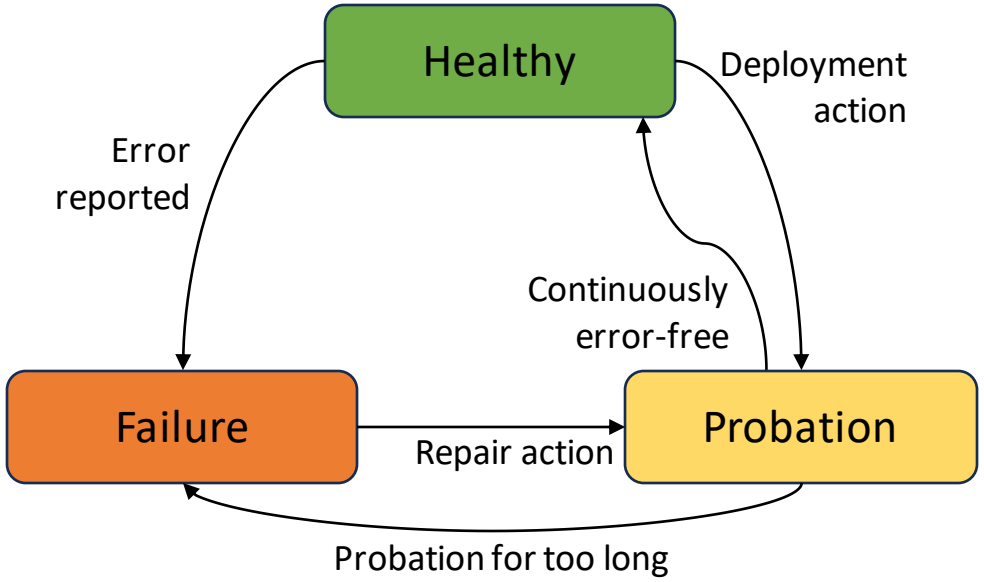
- Deployment service performs update rollouts
  - "Kicks" deployment agents on servers that need to update to a new version
  - Updates a fraction of servers at a time
  - If an update fails, automatically roll back to previous version

Autopilot components – Failure repair



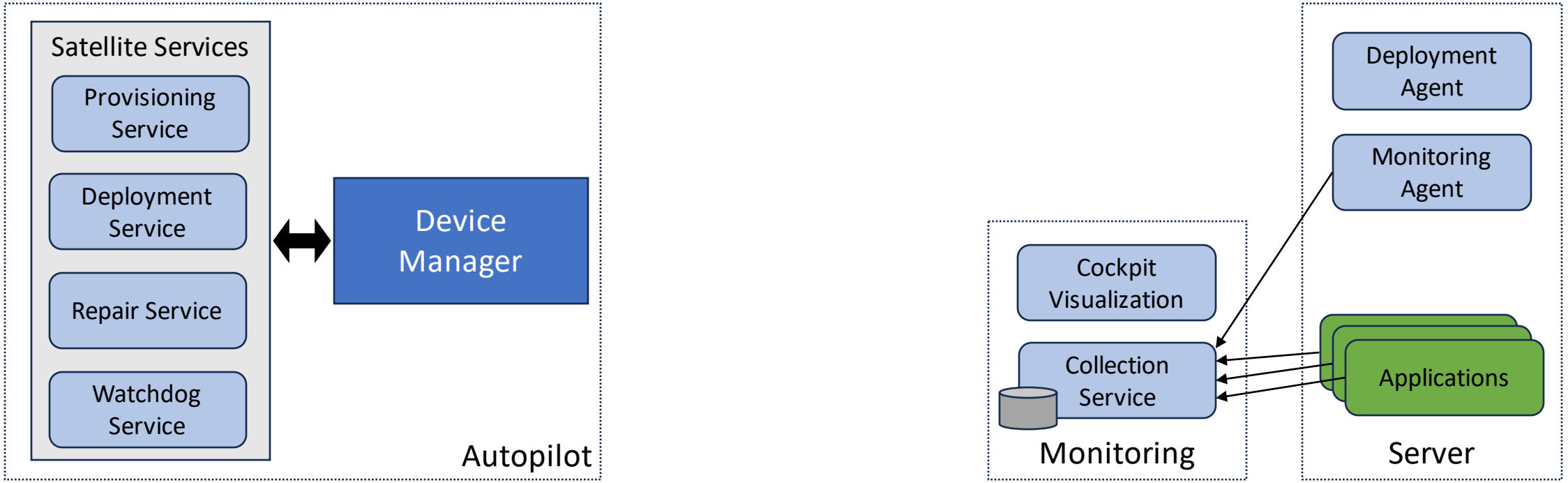
- Failures are detected and repaired automatically
  - Watchdogs probe for monitoring attributes
  - Remediation actions are just reboot, reimage, and replace
  - Remediation granularity is coarse: server or switch

Failure recovery state machine



- Watchdogs can return OK, warning, or error
- Warnings do not trigger human action
  - Applications should try to recover automatically

Autopilot components – Monitoring



- Pervasive monitoring accessible through a database
  - Database facilitates statistical analysis
  - Servers run monitoring agent
  - Applications built to report monitoring metrics to collection service

Other Autopilot modules on servers

- Network is configured via Active Directory
- Filesync service to copy files
  - Logging and better control of transfer schedule and transfer rates
- Application manager to launch necessary applications
  - Assumes applications can be terminated by force

Applications can add specific functionality

- Applications can rely on Autopilot for monitoring and repair
- Applications with more stringent requirements can extend Autopilot
  - Applications may require faster failover
  - Applications may be able to use weak consistency or outdated information

- Example: Windows Live Search
  - Local database allows application to continue operating even if Autopilot fails
  - Load balancing built into the application itself

Evolution of Autopilot

- Constant bug-fixing and engineering improvements
- Easier bootstrapping of an Autopilot cluster
- Better support for applications
  - Initially, Autopilot focused on complex vertical applications
  - Support for microservices and lightweight applications
  - Support for distributed applications (across clusters)

Failures encountered

- Networking hardware will malfunction and corrupt data
- Files will get corrupted, checksums are crucial
  - TCP/IP checksums are weak, need stronger end-to-end verification
- Computers will experience performance degradation
  - System needs to monitor, detect, and repair these non-stop failures
- Throttling and load shedding are crucial
  - Need to be able to distinguish between overload and failure
    - Removing overloaded servers from service will aggravate the problem
  - Prevent cascading failures