WELL-ARCHITECTED FRAMEWORKS (RECORDED)

WELL-ARCHITECTED FRAMEWORK

Review of the well architected framework concepts and application modernization.

The goal of this framework is to enable customers to:

- Assess and improve their architectures
- Better understand the business impact of their design decisions

It provides a set of questions developed by AWS experts to helps customers think critically about their architecture.

And, is applicable to any cloud

It asks, "Does your infrastructure follow best practices?"

THE WELL-ARCHITECTED FRAMEWORK

The Well-Architected Framework does not provide:

- Implementation details
- Architectural patterns
- Relevant case studies

However, it does provide:

- Questions centered on critically understanding architectural decisions
- Services and solutions relevant to each question
- References to relevant resources

PILLARS OF THE WELL-ARCHITECTED FRAMEWORK

Security

Protect and monitor systems.

Reliability

Recover from failure and mitigate disruption.

Performance Efficiency

Use resources sparingly.

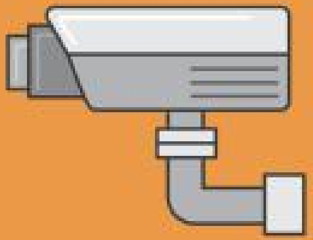
Cost Optimization

Eliminate unneeded expense.

SECURITY

Security

Protect and monitor systems



- The ability to protect:
  - Information
  - Systems
  - Assets
- While delivering business value through:
  - Risk assessments
  - Mitigation strategies

RELIABILITY

Reliability

Recover from failure and mitigate disruption.

The ability of a system to:

- Recover from infrastructure or service failures
- Dynamically acquire computing resources to meet demand
- Mitigate disruptions such as:
  - Misconfigurations
  - Transient network issues

PERFORMANCE EFFICIENCY

Performance Efficiency

Use resources sparingly.

The ability to:

- Use computing resources efficiently to meet system requirements
- Maintain that efficiency as demand changes and technologies evolve

COST OPTIMIZATION

Cost Optimization

Eliminate unneeded expense.

The ability to avoid or eliminate:

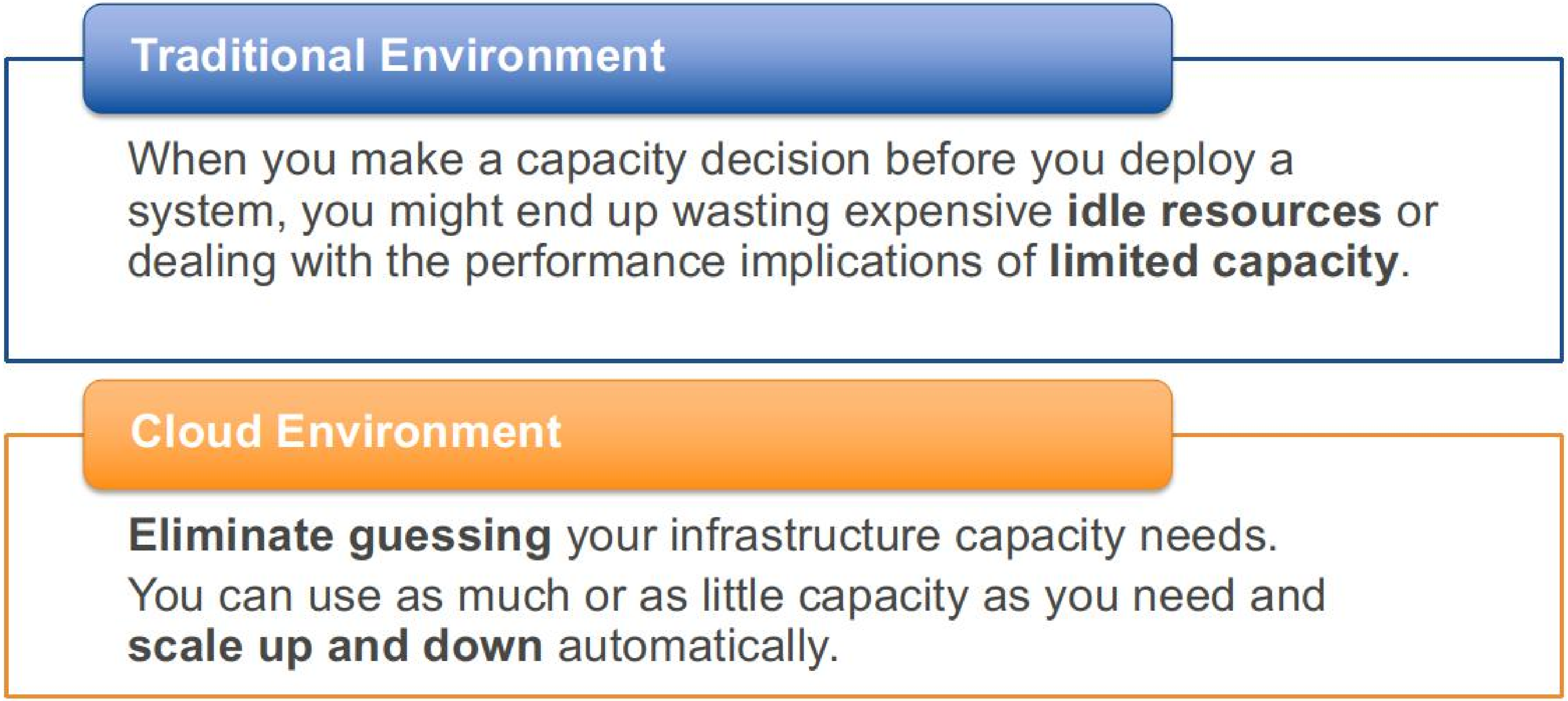
- Unneeded cost
- Suboptimal resources

WELL-ARCHITECTED DESIGN PRINCIPLES

The Well-Architected Framework also identifies a set of general design principles to facilitate good design in the cloud:

- Stop guessing your capacity needs.
- Test systems at production scale.
- Lower the risk of architectural change.
- Automate to make experimentation easier.
- Allow for evolutionary architectures.

STOP GUESSING YOUR CAPACITY NEEDS



Traditional Environment

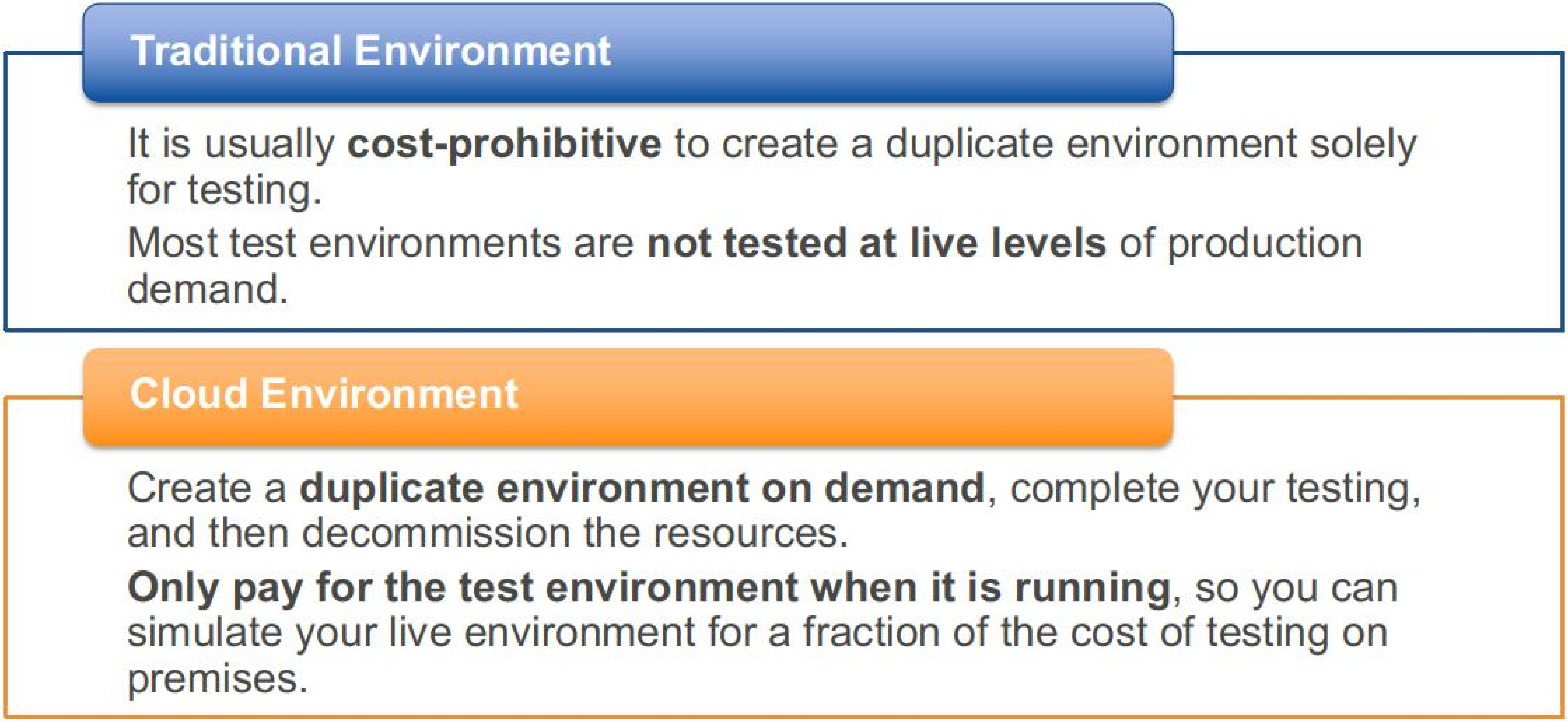
When you make a capacity decision before you deploy a system, you might end up wasting expensive **idle resources** or dealing with the performance implications of **limited capacity**.

Cloud Environment

Eliminate guessing your infrastructure capacity needs.

You can use as much or as little capacity as you need and **scale up and down** automatically.

TEST SYSTEMS AT PRODUCTION SCALE



Traditional Environment

It is usually **cost-prohibitive** to create a duplicate environment solely for testing.

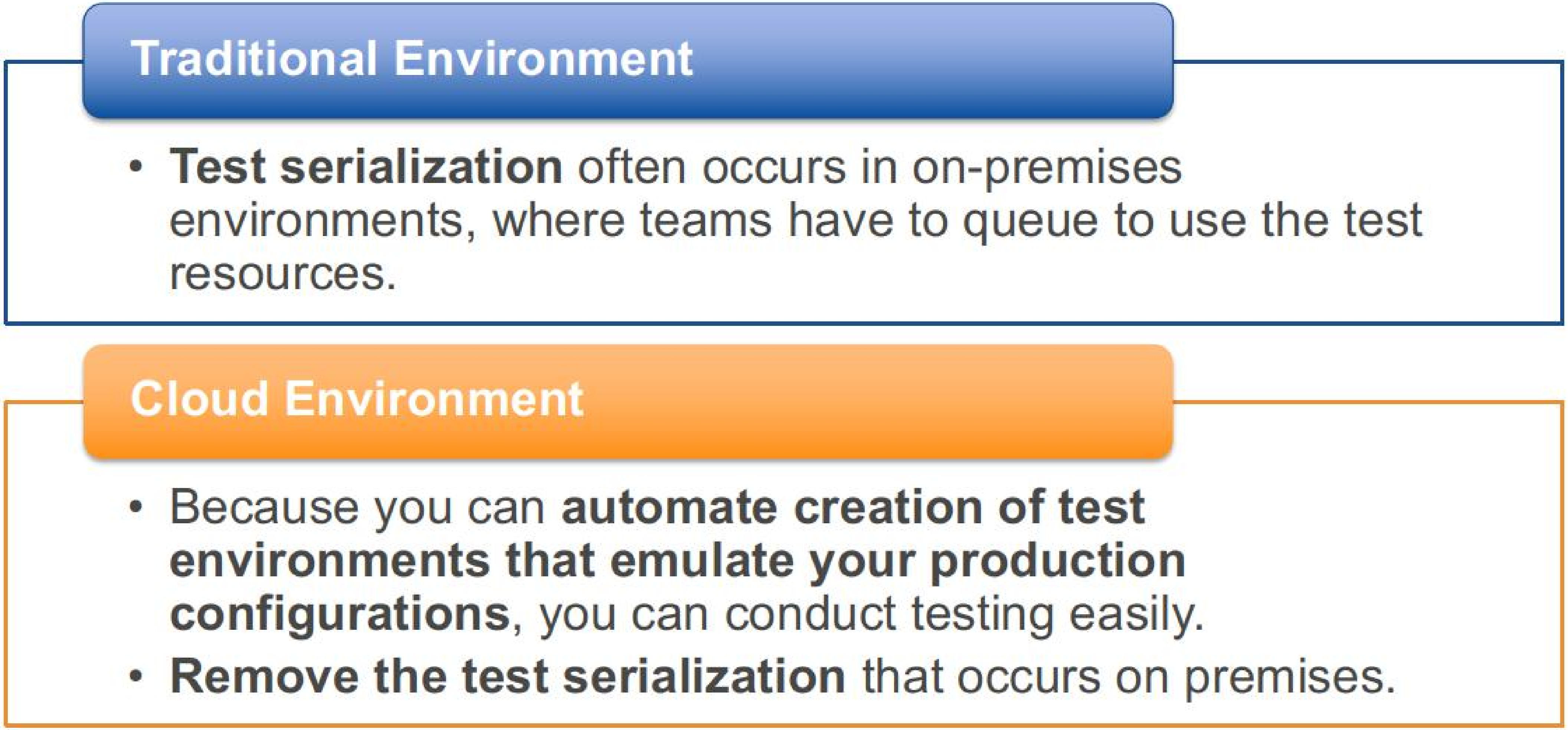
Most test environments are **not tested at live levels** of production demand.

Cloud Environment

Create a **duplicate environment on demand**, complete your testing, and then decommission the resources.

Only pay for the test environment when it is running, so you can simulate your live environment for a fraction of the cost of testing on premises.

LOWER THE RISK OF ARCHITECTURAL CHANGE



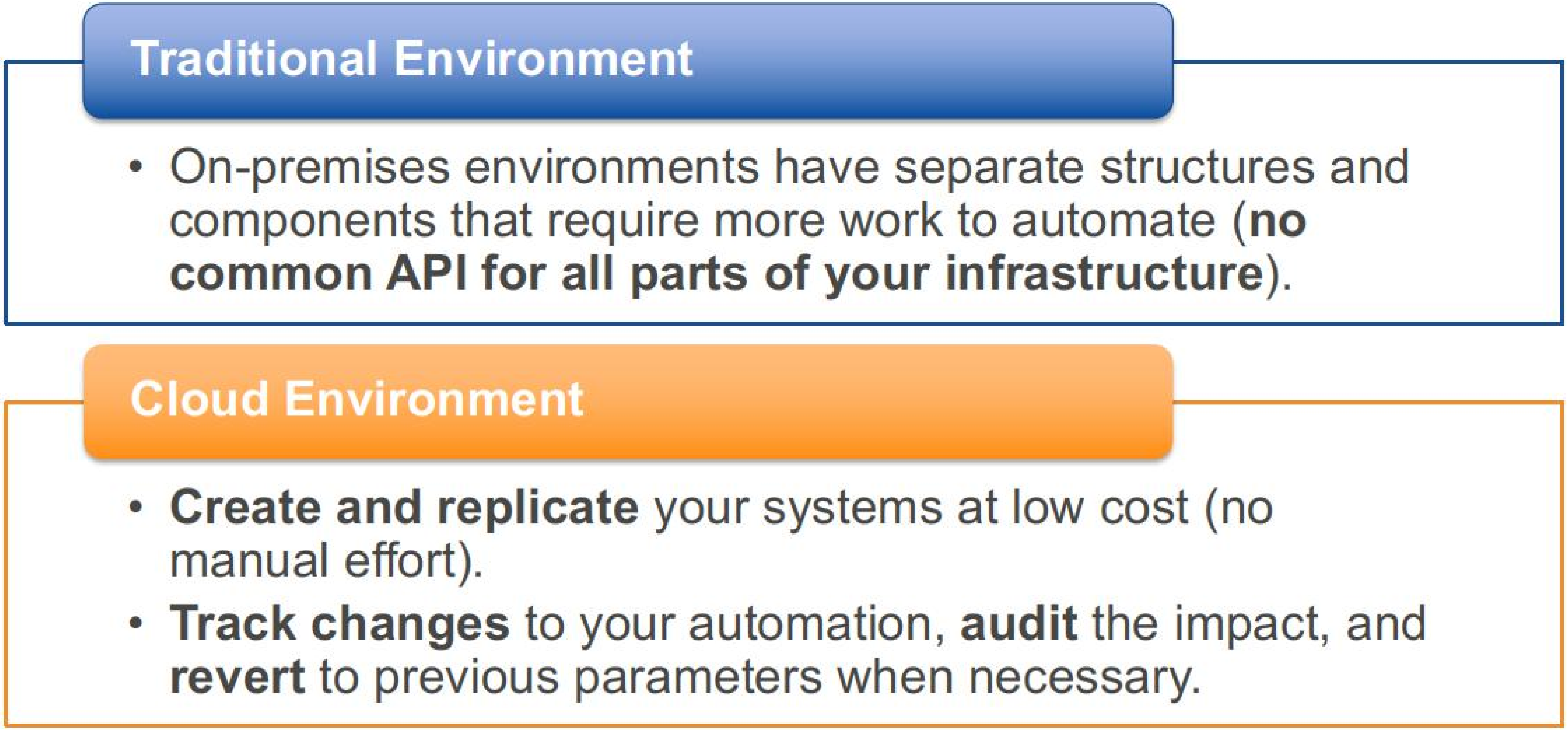
Traditional Environment

- **Test serialization** often occurs in on-premises environments, where teams have to queue to use the test resources.

Cloud Environment

- Because you can **automate creation of test environments that emulate your production configurations**, you can conduct testing easily.
- **Remove the test serialization** that occurs on premises.

AUTOMATE TO MAKE EXPERIMENTATION EASIER



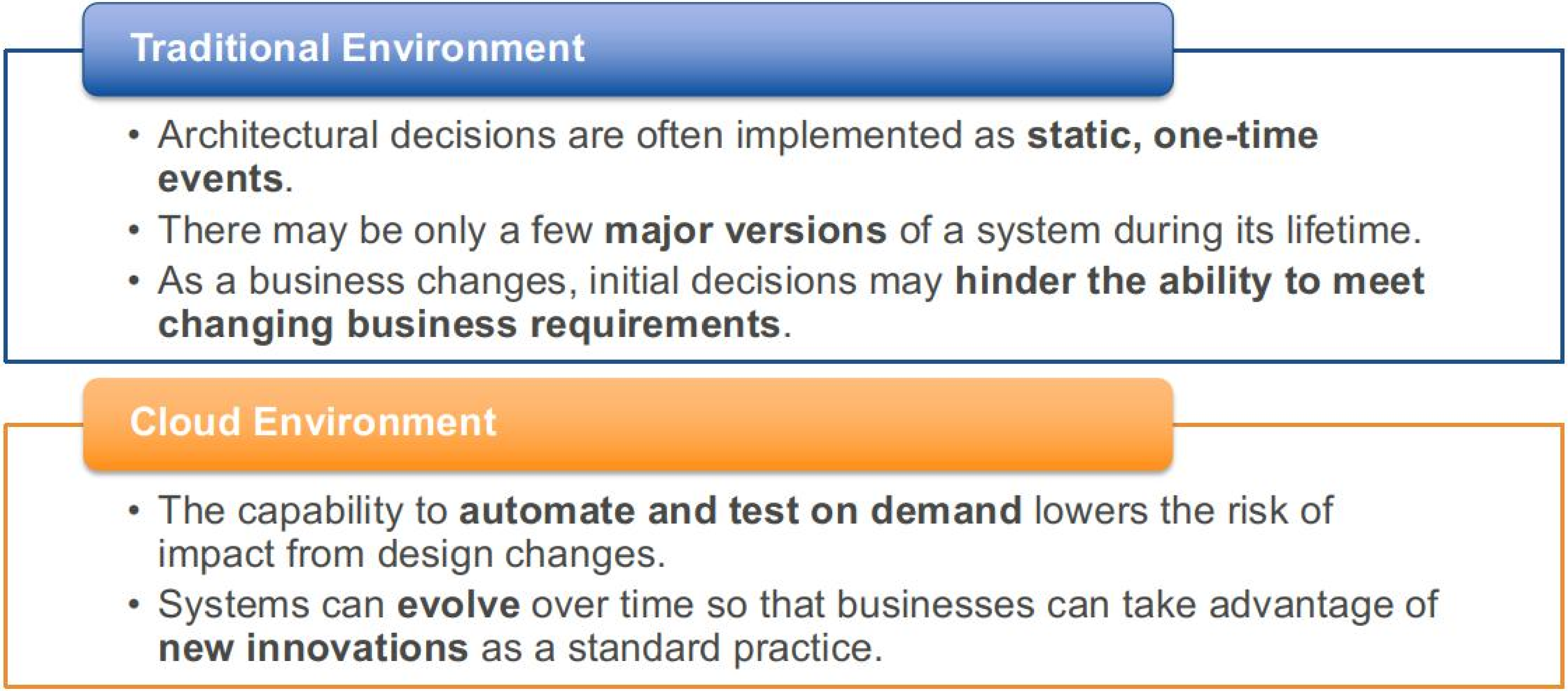
Traditional Environment

- On-premises environments have separate structures and components that require more work to automate (**no common API for all parts of your infrastructure**).

Cloud Environment

- **Create and replicate** your systems at low cost (no manual effort).
- **Track changes** to your automation, **audit** the impact, and **revert** to previous parameters when necessary.

ALLOW FOR EVOLUTIONARY ARCHITECTURES



Traditional Environment

- Architectural decisions are often implemented as **static, one-time events**.
- There may be only a few **major versions** of a system during its lifetime.
- As a business changes, initial decisions may **hinder the ability to meet changing business requirements**.

Cloud Environment

- The capability to **automate and test on demand** lowers the risk of impact from design changes.
- Systems can **evolve** over time so that businesses can take advantage of **new innovations** as a standard practice.

CONGRATS ON COMPLETION