
# What is CloudFormation drift detection and why is it useful? Follow-up: How would you handle configuration drift in production.

**CloudFormation Drift Detection**

Think of drift detection like a security guard who checks if anything in your house has been moved or changed without your permission.

**Why Does Drift Happen in AWS?**

**Common scenarios:**

- A developer manually changes an EC2 instance size through the AWS console
- Someone adds a new security group rule directly (not through CloudFormation)
- A team member modifies database settings using the AWS CLI
- Automated scaling changes instance configurations

**Real-world example:** You deploy a web server using CloudFormation with a t2.micro instance. Later, during a busy period, someone manually upgrades it to t2.large through the console. Your CloudFormation template still says t2.micro, but the actual server is t2.large.

**Why is Drift Detection Useful?**

**1. Maintains Control**

- You always know what's actually running vs. what you planned
- Prevents surprises during the next deployment

**2. Ensures Consistency**

- Your development and production environments stay identical
- Reduces "it works on my machine" problems

**3. Security and Compliance**

- Detects unauthorized changes that might create security holes
- Helps with audit requirements

**4. Cost Management**

- Catches instances that were manually upgraded (and cost more)
- Prevents resource sprawl

**How to Handle Configuration Drift in Production**

**Step 1: Detect the Drift**

**Easy way to check:**

1. Go to your CloudFormation stack in the AWS console
2. Click "Stack actions" → "Detect drift"
3. Wait for the scan to complete
4. Review the results

**What you'll see:**

- **IN_SYNC**: Everything matches your template ✅
- **DRIFTED**: Something was changed manually ⚠️
- **DELETED**: A resource was removed outside CloudFormation ❌
- **NOT_CHECKED**: Resource type doesn't support drift detection

**Step 2: Investigate What Changed**

**Questions to ask:**

- What exactly was changed?
- Who made the change and when?
- Was it an emergency fix?
- Is the change needed permanently?

**Example investigation:**

Resource: WebServerInstance
Expected: t2.micro
Actual: t2.large
Change: Instance type was manually modified

**Step 3: Decide How to Fix It**

You have three main options:

**Option A: Accept the Change (Update Your Template)**

- Best when: The manual change was intentional and should be permanent
- Action: Update your CloudFormation template to match the current state
- Example: If the larger instance is needed for performance, update the template

**Option B: Revert the Change (Restore Original)**

- Best when: The change was accidental or temporary
- Action: Update the stack to restore the original configuration
- Example: Someone accidentally changed settings during troubleshooting

**Option C: Import the Current State**

- Best when: You want to keep current changes but bring them under CloudFormation control
- Action: Use CloudFormation import to manage the drifted resources

**Step 4: Implement the Fix**

**For Option A (Update Template):**

yaml

```
# Before (in template)
WebServer:
  Type: AWS::EC2::Instance
  Properties:
    InstanceType: t2.micro

# After (updated template)
WebServer:
```

```
  Type: AWS::EC2::Instance
  Properties:
    InstanceType: t2.large  # Updated to match reality
```

**For Option B (Revert Changes):**

- Simply run a CloudFormation stack update
- The stack will change the resource back to match your template

**Step 5: Prevent Future Drift**

**Best Practices:**

**1. Team Education**

- Teach your team to make changes through CloudFormation, not manually
- Create clear processes for emergency changes

**2. Regular Monitoring**

- Set up automated drift detection (weekly or monthly)
- Use AWS Config to monitor configuration changes

**3. Change Management Process**

Emergency Change → Document it → Update Template → Deploy Template

**4. Use CloudFormation Stack Policies**

- Protect critical resources from accidental changes
- Require specific permissions for manual modifications

**Simple Workflow for Beginners**

**When you find drift:**

1. **Don't Panic** - Drift is common and fixable
2. **Understand What Changed** - Look at the drift detection results
3. **Ask "Is This Change Good?"**
    - Yes → Update your template to match
    - No → Revert by running a stack update
4. **Document the Incident** - Learn from what happened
5. **Prevent It Next Time** - Improve your processes

**What is one security-related benefit of drift detection?**

A. Encrypts all manual changes automatically
B. Identifies unauthorized changes that could create vulnerabilities
C. Blocks public access to S3
D. Prevents IAM user creation outside the template
**Answer:** B

---

**Which of the following is a real-world example of configuration drift?**

A. A stack update failed due to syntax error
B. A t2.micro instance was changed to t2.large manually
C. An S3 bucket was deleted using CloudFormation
D. A Lambda function failed due to a missing layer
**Answer:** B

---