Devops

Virtual Machines

The Land/House Analogy

Inefficient Use (Traditional Approach)

- You have **1 acre of land**
- You build a house using only **0.5 acre**
- **0.5 acre remains unused** = **WASTE/INEFFICIENCY**

Efficient Use (Virtualization Approach)

- Build **another house on the unused 0.5 acre**
- Rent it out to someone else
- **Both properties utilized** = **EFFICIENCY**
- You get benefits while staying on your land

Applying to Software/Servers

Problem: Inefficient Server Usage (Before Virtualization)

Scenario at example.com:

- Company buys **5 physical servers from HP**
- Each server has: **100GB storage, 100 cores CPU**

Server allocation:

Server 1 (P1): 100GB, 100 cores → Using only 20GB, 30 cores

Server 2 (P2): 100GB, 100 cores → Using only 15GB, 25 cores

Server 3: 100GB, 100 cores → Using only 10GB, 20 cores

Server 4: 100GB, 100 cores → Using only 25GB, 40 cores

Server 5 (Prod): 100GB, 100 cores → Using only 5GB, 10 cores (45GB wasted!)

```

**Result:**
```

- **Massive waste** of CPU and storage across all servers
- **High costs** for underutilized hardware
- **INEFFICIENCY**

Solution: Virtualization

What is Virtualization?

Instead of running one OS per physical server, we:

1. **Buy ONE powerful physical server** (100GB, 100 cores)
2. **Install Hypervisor** (virtualization software layer)
3. **Create multiple Virtual Machines (VMs)** on that single server
4. Each VM acts as an **independent computer** with its own:
   - Memory
   - CPU
   - Storage
   - Operating System

Architecture:

```
Physical Server (100GB, 100 cores)
        ↓
   HYPERVISOR (VMware, Xen, Hyper-V, KVM)
        ↓
  Logical Isolation
```

↓

VM1 (10GB, 12 cores) | VM2 (10GB, 12 cores) | VM3 (10GB, 12 cores) | VM4 (10GB, 12 cores) | VM5 (10GB, 12 cores)

Key Benefits:

✅ **Efficiency** - One physical server now hosts 5 VMs instead of needing 5 physical servers

✅ **Isolation** - VM1 is completely independent from VM2, VM3, etc.

✅ **Cost Savings** - Reduce hardware costs by ~80%

✅ **Flexibility** - Easy to create, delete, or resize VMs

✅ **Better Resource Utilization** - No wasted CPU/memory

Real-World Example: Amazon AWS

How AWS Works:

1. **Amazon builds massive data centers** in regions (Mumbai, Singapore, Ohio, etc.)
2. **Millions of physical servers** installed in these data centers
3. **Each physical server has hypervisor installed** (likely Xen/KVM)
4. **When you request a VM:**
    - Select **Region** (e.g., Mumbai)
    - Choose **VM specifications** (10GB RAM, 12 cores)
    - AWS hypervisor **creates a VM** on one of their physical servers
    - You get an **EC2 instance** (AWS's name for VM)
5. **You pay only for what you use** (e.g., $60/month for your VM specs)

AWS Regions (from your diagram):

- **AWS Singapore**
- **AWS Mumbai**
- **AWS Ohio**

Each region has **data centers** with physical servers running hypervisors creating VMs for customers worldwide.

Popular Hypervisors

1. **VMware** (ESXi) - Enterprise standard
2. **Xen** - Used by AWS
3. **KVM** - Linux-based, open source
4. **Hyper-V** - Microsoft's hypervisor
5. **VirtualBox** - Desktop virtualization

Key Concept: Hypervisor

The **hypervisor** is the magic software that:

- Sits between physical hardware and VMs
- **Logically separates** resources
- Allocates CPU, memory, storage to each VM
- Ensures VMs don't interfere with each other
- Makes one physical server act like many independent computers

---

Summary

Before Virtualization:

- 5 physical servers, mostly underutilized
- High cost, low efficiency

After Virtualization:

- 1 physical server with hypervisor
- 5 VMs running efficiently
- Lower cost, high efficiency
- Foundation for **cloud computing** (AWS, Azure, GCP)

This is why virtualization revolutionized IT infrastructure and made cloud computing possible! 🚀

The Automation Problem

Manual vs Automated Approach

Scenario: You receive **100 requests** for EC2 instances

Manual Approach (Bad):

- Developer X logs into AWS Console
- Clicks through UI 100 times
- Creates each EC2 instance manually
- ❌ Time-consuming
- ❌ Error-prone
- ❌ NOT SCALABLE

Automated Approach (Good):

- Write a **script once**
- Script calls **AWS EC2 API**
- Creates 100 instances automatically
- ✅ Fast
- ✅ Consistent
- ✅ Scalable

How AWS API Works

AWS EC2 API Architecture:

User Script → AWS EC2 API → Validation → EC2 Instance Created

```

**API Requirements:**

1. **Valid format** - Request matches expected structure

2. **Authenticated** - User identity verified

3. **Authorized** - User has permissions
```

If all criteria met → API returns EC2 instance

Tools to Automate EC2 Creation

5 Ways to Talk to AWS EC2 API:

Tool	Description	Use Case
1. AWS CLI	Command-line interface	Quick scripts, one-off commands
2. AWS SDK (Boto3)	Python library for AWS API	Complex automation, custom apps
3. AWS CloudFormation (CFT)	AWS-native infrastructure as code	AWS-only deployments
4. AWS CDK	Code-based infrastructure (TypeScript, Python)	Developer-friendly IaC
5. Terraform	Open-source, multi-cloud IaC	**Hybrid/multi-cloud environments**

When to Use Terraform:

Use Terraform for **hybrid cloud models**:

- VM1 in AWS
- VM2 in Azure
- VM3 in Google Cloud

Terraform manages all from **one tool** ✅

Practical: Creating EC2 Instance Manually

Step-by-Step Process:

1. Launch EC2 Instance

```

AWS Console → EC2 Dashboard → Launch Instance

**Configuration:**

- **Name:** Give your instance a name (e.g., "my-first-ec2")
- **OS (AMI):** Ubuntu (or any Linux distribution)
- **Instance Type:** t2.micro (Free tier eligible)
    - Limited resources (1 vCPU, 1GB RAM)
    - Good for learning/testing

**2. Create Key Pair (Important!)**

**What is Key Pair?**

- Used for **SSH authentication**
- Allows you to **securely login** to EC2 instance

**Steps:**

1. Click "Create new key pair"
2. **Name:** e.g., "my-ec2-key"
3. **Type:** RSA
4. **Format:** .pem (for Linux/Mac) or .ppk (for PuTTY/Windows)
5. Click **Create**
6. **File downloads** to your local machine (e.g., my-ec2-key.pem)

⚠️ **IMPORTANT:** Keep this file safe! You can't download it again.
```

3. Launch Instance

Click **"Launch Instance"**

✅ Instance is now **running**!

Connecting to EC2 Instance via MobaXterm

Step 1: Get EC2 IP Address

1. Go to EC2 Dashboard
2. Select your instance
3. **Copy Public IPv4 address** (e.g., 54.123.45.67)

Step 2: Configure MobaXterm

Open MobaXterm:

1. Click **"Session"**
2. Select **"SSH"**

Basic Settings:

- **Remote host:** Paste IP address (54.123.45.67)
- **Port:** 22 (default SSH port)
- **Username:** ubuntu (for Ubuntu AMI) or ec2-user (for Amazon Linux)

Advanced SSH Settings:

1. Click **"Advanced SSH settings"** tab
2. Check ✅ **"Use private key"**
3. Browse and select your .pem file (e.g., my-ec2-key.pem)
4. Click **"OK"**

Step 3: Login

MobaXterm will connect to your EC2 instance!

You should see:

bash

ubuntu@ip-172-31-12-34:~$

🎉 **You're now logged into your EC2 instance!**