# Architecture in a Virtual Private Cloud (VPC)

When designing your infrastructure in the public Cloud, we recommend using a Virtual Private Cloud (VPC) to allow isolation and network control within your infrastructure.

To ensure security while maintaining service accessibility, we usually design a VPC in the same way we design n-tier applications.



**Related Pages**

- About VPCs
- Reducing Latency

## Designing Cloud Services

When designing your Cloud services, we recommend using the following guidelines and best practices:

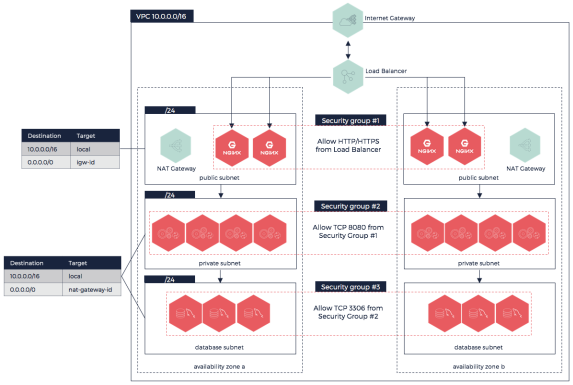
- Design for failure, and create self-healing application environments.
- Always design applications with instances in at least two Availability Zones. For more information, see About Regions, Endpoints, and Availability Zones.
- Guarantee that you have "reserved" capacity in the event of an emergency by purchasing reserved instances in a designated recovery Availability Zone. For more information, see About Instances > Reserved Instances.
- Rigorously test to find single points of failure and apply high availability.
- Use External IP Addresses (EIPs) for fail over to "stand-by" instances when auto-scaling and load balancing are not available. For more information, see About EIPs.
- Have a disaster recovery and backup strategy that:
  - Uses multiple Regions,
  - Maintains up-to-date OMIs (and copies OMIs from one Region to another),
  - Copies BSU snapshots to other Regions (and uses CRON jobs that take snapshots of BSU). For more information, see Tutorial: Copying a Snapshot to a Different Region,
  - Automates everything in order to easily redeploy resources in the event of a disaster,
  - Uses bootstrapping to quickly bring up new instances with minimal configuration and allows for "generic" OMIs.
- Decouple application components using Message Broker software when possible.
- "Throw away" old or broken images.
- Overuse telemetry and metrics to monitor infrastructure changes and health.
- Use Multipart upload for OUTSCALE Object Storage (OOS) objects over 100 MiB. For more information, see Using Multipart Upload.
- Protect your data in transit using HTTPS/SSL endpoints.
- Protect your data at rest using encrypted file systems.
- Connect to instances inside a VPC using a bastion host or a VPN connection.
- (Available soon) Use EIM roles on FCU instances instead of using API keys, and never store API keys on an OMI.

## Implementing a 3-Tier Architecture with Public and Private Subnets

For the purpose of this example, we consider that almost all platforms are designed and deployed with two types of services:

- Internal/private services: Only used by your platform to serve a higher purpose, also called backends.
- External/public services: Used by the end user as an interface, also called frontends (we call "External/Public" a machine that has a public IP address attached).

You need to define public and private services before moving forward on the implementation of your Cloud infrastructure. The public services will be accessible from outside the VPC by the users, while the private services will not be exposed outside and will be only accessible by other services within the VPC.




For security, traceability and simplicity purposes, we follow these rules:

- One service per instance,
- One service per subnet,
- One security group per subnet,
- One route table per subnet.

The demo platform is composed of:

- NGINX fronts,
- Intel application handling the request and the load from the NGINX,
- Database instances for data storage and queried by the Intels.

Each layer only needs to communicate with the one above and the one below. The security groups are set accordingly, and you can see that we respected the rule "one service per instance, one service per subnet, one service per security group".

# Example Using Boto2

**Network setup**

```
from boto.ec2.regioninfo import EC2RegionInfo
from boto.vpc import VPCConnection

your_ak, your_sk = "XXXXXAAAXXXXXX", "ZZZZZZBBBBBBZZZZZZBBBBBBZZZZZZZ"
outscale_endpoint = EC2RegionInfo(endpoint="fcu.eu-west-2.outscale.
com")    # you can change the region
outscale_fcu = VPCConnection( aws_access_key_id = your_ak,
aws_secret_access_key = your_sk, region = outscale_endpoint)




# Setup network
vpc, sub_front, sub_intel = setup_vpc(vpc_cidr='10.0.0.0/16')
sub_db = outscale_fcu.create_subnet(vpc_id=vpc.id, cidr_block='10.0.3.0
/24')
rt_db = outscale_fcu.create_route_table(vpc_id=vpc.id)
outscale_fcu.associate_route_table(route_table_id=rt_db.id,
subnet_id=sub_db.id)

# Setup security
sg_nginx = outscale_fcu.create_security_group(name='nginx',
description='Web front security group', vpc_id=vpc.id)
sg_intel = outscale_fcu.create_security_group(name='intel',
description='Intel security group', vpc_id=vpc.id)
sg_db = outscale_fcu.create_security_group(name='db',
description='Database security group', vpc_id=vpc.id)

# Now you need to allow traffic between applications
for port in [80, 443]:
    sg_nginx.authorize(ip_protocol='tcp', from_port=port, to_port=port,
cidr_ip='0.0.0.0/0')
sg_intel.authorize(ip_protocol='tcp', from_port=8080, to_port=8080,
src_group=sg_nginx)
sg_db.authorize(ip_protocol='tcp', from_port=3306, to_port=3306,
src_group=sg_intel)
```

This example suggests that we handled the SSL termination on the NGINX, that we use classic HTTP port for our intel (8080) and that we implemented MySQL databases listening on port 3306.

We used security groups to access security groups instead of using CIDR of subnets, even if in our case, both approaches offer similar results. For more information, see About Security Groups.

This architecture can be applied to more complex applications, and enforces the state of the art in terms of security and network control.

**AWS**™ and **Amazon Web Services**™ are trademarks of Amazon Technologies, Inc or its affiliates in the United States and/or other countries.