**Advanced-EC2**

**Bootstrapping EC2 using User Data**

Bootstrapping is a process where scripts or other config steps can be run when an instance is first launched. This allows an instance to be brought to service in a particular configured state.

In systems automation, bootstrapping allows the system to self-configure. In AWS this is **EC2 Build Automation**.

This could perform some software installs and post install configs.

Bootstrapping is done using **user data** and is injected into the instance in the same way that meta-data is. It is accessed using the meta-data IP.

<http://169.254.169.254/latest/user-data>

Anything you pass in is executed by the instance OS only once on launch!

EC2 doesn't validate the user data. You can tell EC2 to pass in trash data and the data will be injected. The OS needs to understand the user data.

**Bootstrapping Architecture**

An AMI is used to launch an EC2 instance in the usual way to create an EBS volume that is attached to the EC2 instance. This is based on the block mapping inside the AMI.

Now the EC2 service provides some user data through to the EC2 instance. There is SW within the OS designed to look at the metadata IP for any user data. If it sees any user data, it executes this on launch of that instance.

This is treated like any other script the OS runs. At the end of running the script, the instance will be in:

* Running state and ready for service.
* Bad config but still likely running.
  + The instance will probably still pass its checks.
  + It will not be configured as you expected.

**User Data Key Points**

EC2 doesn't know what the user data contains, it's just a block of data. The user data is not secure, anyone can see what gets passed in. For this reason it is important not to pass passwords or long term credentials.

User data is limited to 16 KB in size. Anything larger than this will need to pass a script to download the larger set of data.

User data can be modified however if you stop the instance, change the user data, then restart the instance. This won't be executed since the instance has already started.

**Boot-Time-To-Service-Time**

How quickly after you launch an instance is it ready for service? This includes the time for EC2 to configure the instance and any software downloads that are needed for the user. When looking at an AMI, this can be measured in minutes.

AMI baking will front load the time needed by configuring as much as possible.

**AWS::CloudFormation::Init**

**cfn-init** is a helper script installed on EC2 OS. This is a simple configuration management system.

* User Data is procedural and run by the OS line by line.
* cfn-init can be procedural, but can also be **desired state**.
  + Can specify particular versions of packages. It will ensure things are configured to that end state.
  + Can manipulate OS groups and users.
  + Can download sources and extract them using authentication.

This is executed as any other command by being passed into the instance as part of the user data and retrieves its directives from the CloudFormation stack and you define this data in the CloudFormation template called AWS::CloudFormation::Init.

**cfn-init explained**

Starts off with a **CloudFormation template**. This has a logical resource within it which is to create an EC2 instance. This has a specific section called Metadata. This then passes in the information passed in as UserData. cfn-init gets variables passed into the user data by CloudFormation.

It knows the desired state and can work towards a final configuration. This can monitor the user data and change things as the EC2 data changes.

**CreationPolicy and Signals**

If you pass in user data, there is no way for CloudFormation to know if the EC2 instance was provisioned properly. It may be marked as complete, but the instance could be broken.

A **CreationPolicy** is something which is added to a logical resource inside a CloudFormation template. You create it and supply a timeout value.

This waits for a signal from the resource itself before moving to a create complete state.

**EC2 Instance Roles**

IAM roles are the best practice ways for services to be granted permissions. EC2 instance roles are roles that an instance can assume and anything running in that instance has the permissions that role grants.

Starts with an IAM role with a permissions policy. EC2 instance role allows the EC2 service to assume that role.

The **instance profile** is the item that allows the permissions to get inside the instance. When you create an instance role in the console, an instance profile is created with the same name.

When IAM roles are assumed, you are provided temporary roles based on the permission assigned to that role. These credentials are passed through instance **meta-data**.

EC2 and the secure token service ensure the credentials never expire.

Key facts

* Credentials are inside meta-data
  + iam/security-credentials/role-name
  + automatically rotated - always valid
  + Resources need to check the meta-data periodically
* Should always use roles compared to storing long term credentials
* CLI tools use role credentials automatically

**AWS System Manager Parameter Store**

Passing secrets into an EC2 instance is bad practice because anyone who has access to the meta-data has access to the secrets.

Parameter store allows for storage of **configuration** and **secrets**

* Strings
* StringList
* SecureString

**Parameter Store:**

* Can store **license codes, database strings, and full configs and passwords**.
* Allows for **hierarchies(using prefixes) and versioning**.
* Can store plaintext and ciphertext.
  + This integrates with **kms** to encrypt passwords.
* Allows for public parameters such as the latest AMI parameter to be stored and referenced for EC2 creating.
* Is a public service so any services needs access to the public sphere or to be an AWS public service.
* Applications, EC2 instances, lambda functions can all request access to parameter store.
* Tied closely to IAM, can use
  + Long term credentials such as access keys.
  + Short term use of IAM roles.

**System and Application Logging on EC2**

CloudWatch and CloudWatch Logs cannot natively capture data inside an instance.

CloudWatch Agent is required for OS visible data. It sends this data into CW For CW to function, it needs configuration and permissions in addition to having the CW agent installed. The CW agent needs to know what information to inject into CW and CW Logs.

The agent also needs some permissions to interact with AWS. This is done with an IAM role as best practice. The IAM role has permissions to interact with CW logs. The IAM role is attached to the instance which provides the instance and anything running on the instance, permissions to manage CW logs.

The data requested is then injected in CW logs. There is one log group for each individual log we want to capture. There is one log stream in each group for each instance that needs management.

We can use parameter store to store the configuration for the CW agent.

**EC2 Placement Groups**

**Cluster Placement**

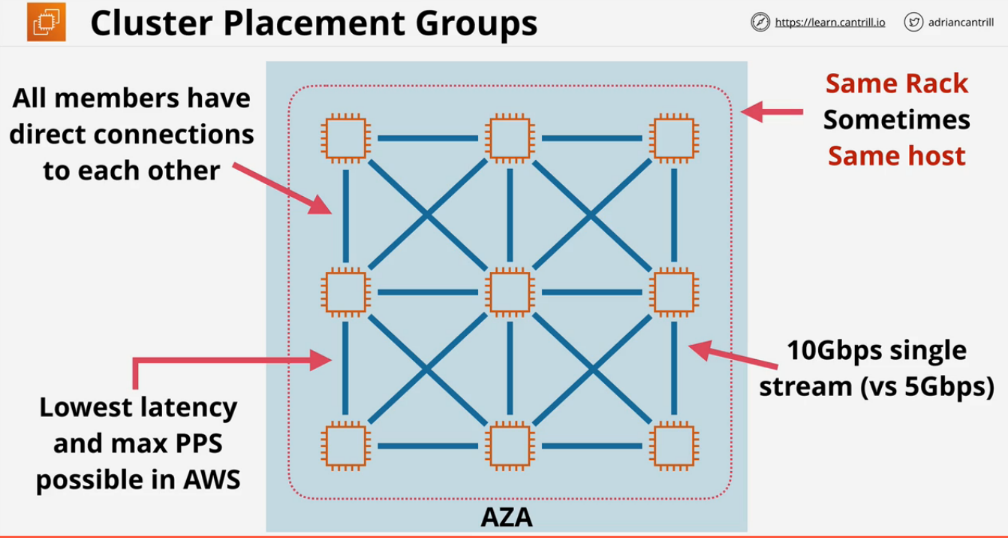
Pack instances close together

Achieves the **highest level of performance available with EC2**.

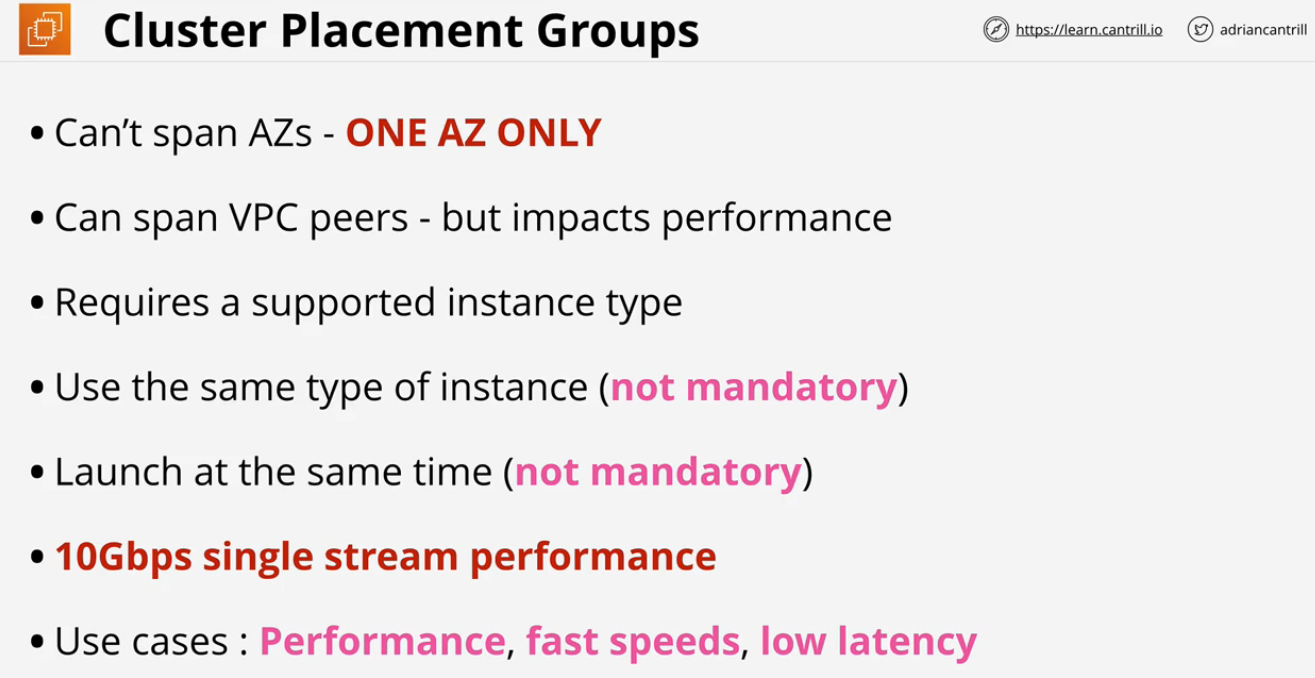
Best practice is to launch all of the instances within that group at the same time. If you launch with 9 instances and AWS places you in a place with capacity for 12, you are now limited in how many you can add.

Cluster placements need to be part of the same AZ. Cluster placement groups are generally the same rack, but they can even be the same EC2 host.

All members have direct connections to each other. They can achieve **10 Gbps single stream** vs **5 Gbps normally**. They also have the **lowest latency** and max **PPS** possible in AWS.

If the hardware fails, the entire cluster will fail.

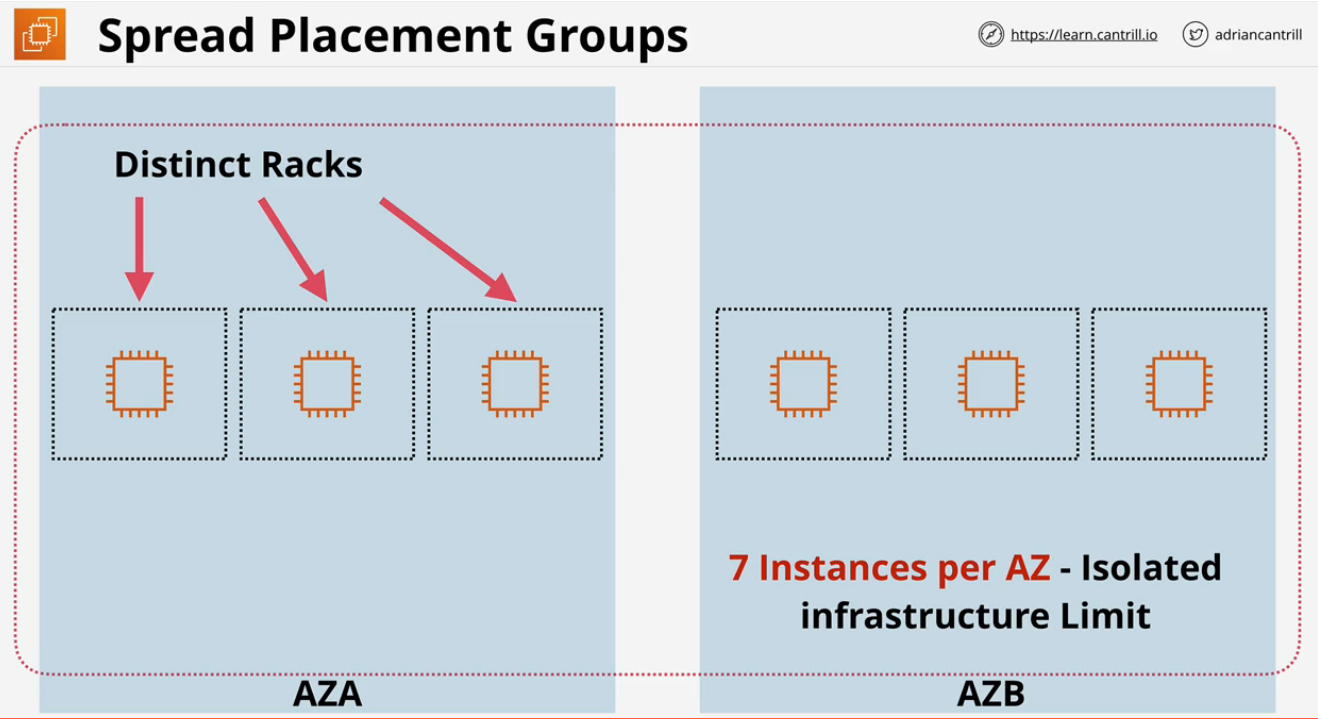
**Cluster Placement Exam PowerUp**

* Clusters can't span AZs. The first AZ used will lock down the cluster.
* They can span VPC peers.
* Requires a supported instance type. <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/placement-groups.html>
* Best practice to use the same type of instance and launch all at once.
* This is the only way to achieve **10Gbps SINGLE stream**, other data metrics assume multiple streams.[EXAM]

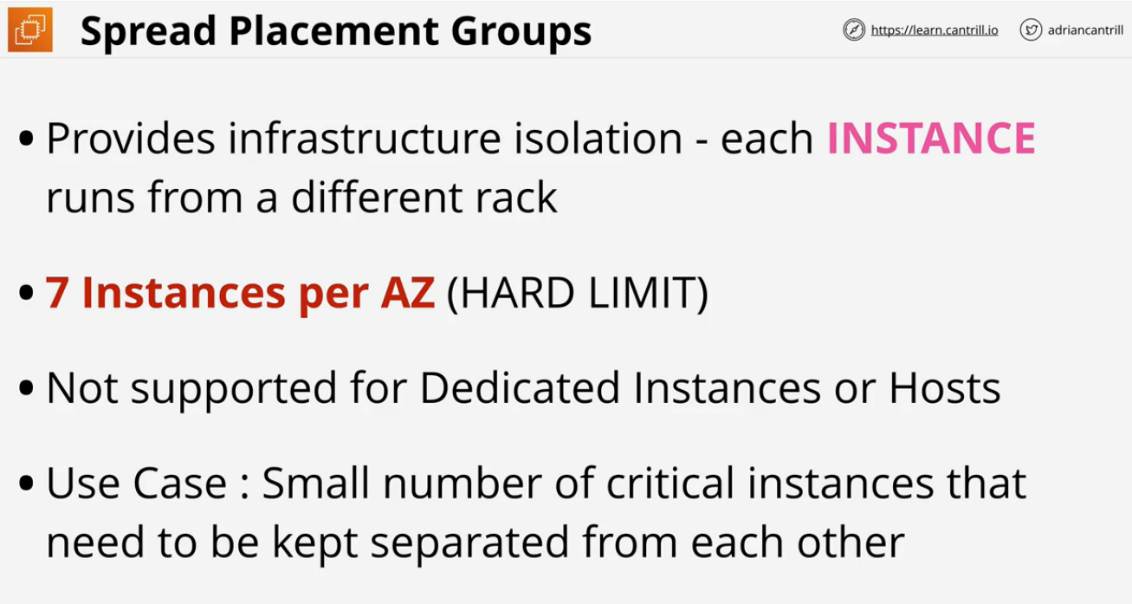
**Spread Placement**

Keep instances separated

This provides the best resilience and availability. Spread groups can span multiple AZs. Instances will be put on distinct racks with their own network or power supply. There is a limit of 7 instances per AZ. The more AZs in a region, the more instances inside a spread placement group.

**Spread Placement Exam PowerUp**

* Provides the highest level of availability and resilience.
  + Each instance by default runs from a different rack.
* 7 instances per AZ is a hard limit.
* Not supported for dedicated instances or hosts.
* Placement is handled by default natively by AWS.

Use case: small number of critical instances that need to be kept separated from each other. Several mirrors of an application, domain controllers, different file server mirrors for a business critical application, different nodes for a large scale Analytics cluster.

**Partition Placement**

Groups of instances spread apart.

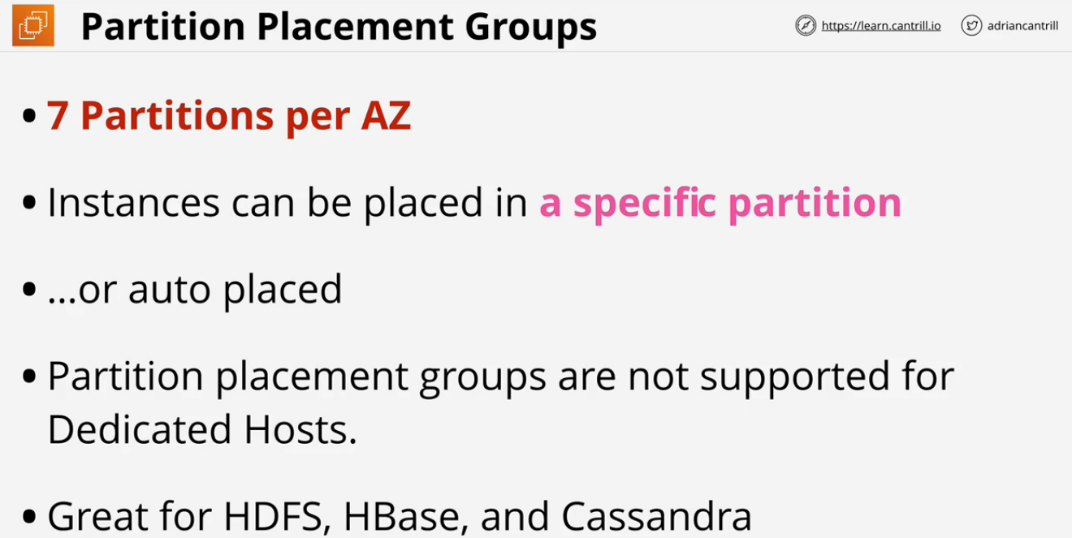
Designed when you have more than 7 instances per availability but you still require the ability to separate those instances into separate areas of fault.

If a problem occurs with one rack's networking or power, it will at most take out one instance.

The main difference is you can launch as many instances in each partition as you desire.

When you launch a partition group, you can allow AWS decide or you can specifically decide.

**Partition Placement Exam PowerUp**

* 7 partitions maximum for each AZ
* Instances can be placed into a specific partition(more control on where you are launching the instance so you get to design your own resiliency), or AWS can pick.
* This is not supported on dedicated hosts.
* Visibility of the Partitions which allows you to pass on the data to topology aware applications like HDFS, HBase, and Cassandra for deciding which instances the database etc to be replicated on to.
* As opposed to Spread placement where placement is handled by default natively by AWS, with Partition user defines which partition the instance is launched in.
* Designed for large scale system where you need to create groupings of instances and have them separated.

**EC2 Dedicated Hosts**

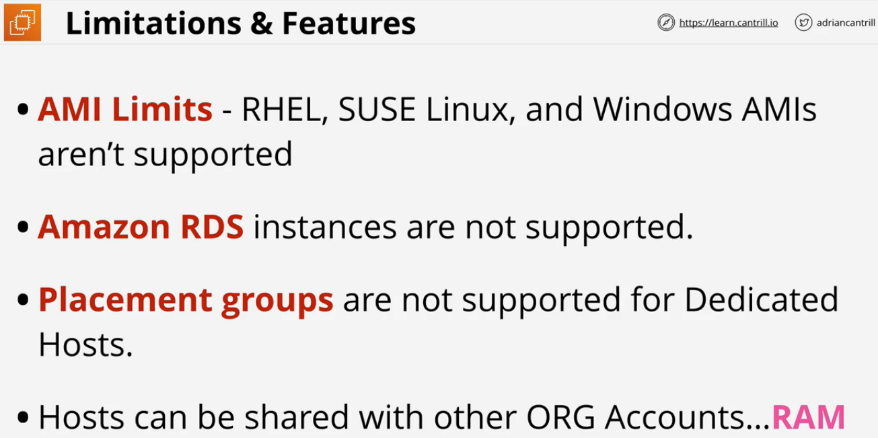
EC2 host allocated to you in its entirety. Pay for the host itself which is designed for a family of instances. There are no instance charges. You can pay for a host on-demand or reservation with 1 or 3 year terms.

The host hardware has physical sockets and cores. This dictates how many instances can be run on the HW.

Hosts are designed for a specific size and family. If you purchase one host, you configure what type of instances you want to run on it. With the older VM system you cannot mix and match. The new Nitro system allows for mixing and matching host size.

Dedicated hosts are EC2 Hosts which support a certain type of instance which are dedicated to your account.

You can pay an on-demand or reserved price for the hosts and then you have no EC2 instance pricing to pay for instances running on these dedicated hosts.

Generally dedicated hosts are used for applications which use physical core/socket licensing

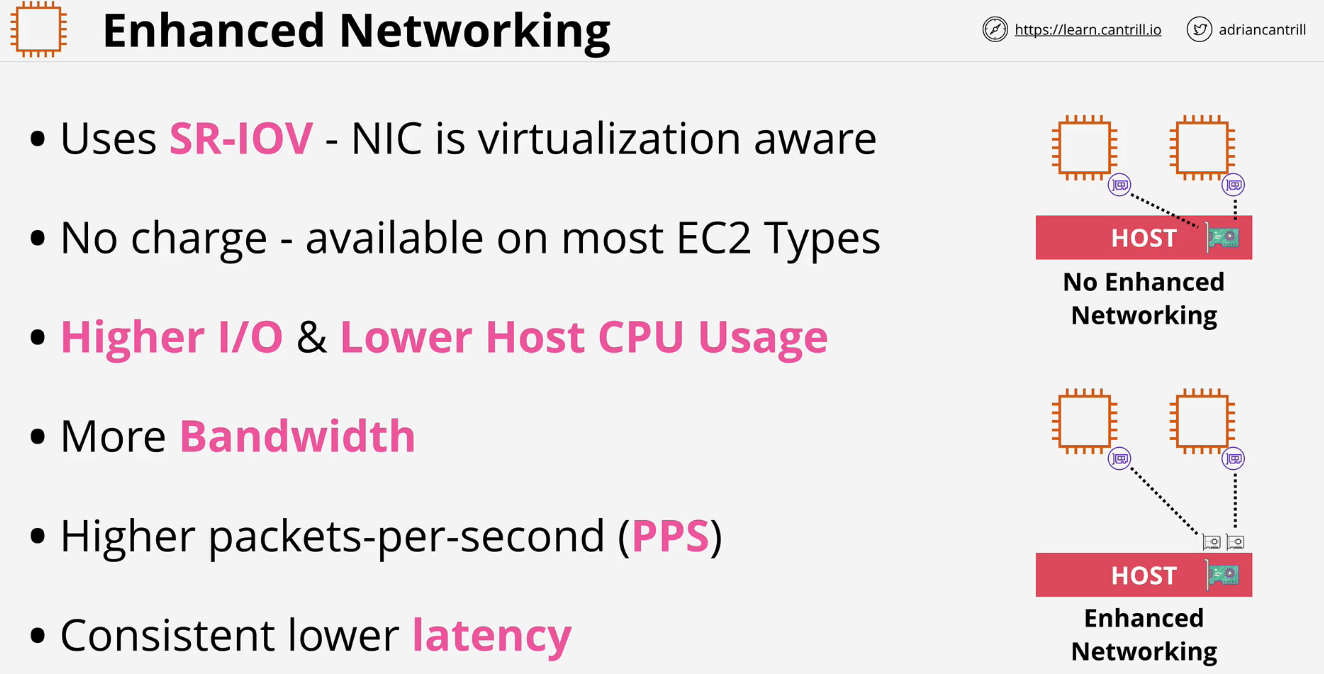
**Dedicated Hosts Limitations**

* AMI Limits, some versions can't be used
* Amazon RDS instances are not supported
* Placement groups are not supported for dedicated hosts.
* Hosts can be shared with other organization accounts using **RAM**
* This is mostly used for licensing problems related to ports.

**Single Root I/O Virtualization**

Enhanced networking uses SR-IOV. The physical network interface is aware of the virtualization. Each instance is given exclusive access to one part of a physical network interface card.

There is no charge for this and is available on most EC2 types. It allows for higher IO and lower host CPU usage This provides more bandwidth and higher packet per seconds. In general this provides lower latency.



**EBS Optimized**

Historically network on EC2 was shared with the same network stack used for both data networking and EBS storage networking.

EBS optimized instance means that some stack optimizations have taken place and dedicated capacity has been provided for that instance for EBS usage.

Most new instances support this and have this enabled by default for no charge.

