Scott Gerring for AWS Switzerland
Posted on 24 Aug 2021

💖 11

AWS Latency Optimization for Trading Applications

#aws

Solutions on AWS that need consistently low latency between components are an interesting class of problem. Cloud platforms virtualize compute and network resources to optimize for more common problems, simultaneously impacting latency. We can readily achieve single digit millisecond latency with solutions built on top of AWS, and for the majority of use cases, this is great, but for some domains, we may want to try and do better.

Despite the virtualization, there are plenty of tools we can reach for to drive our latency down - we just have to look a bit harder for them. A classic use case we often see in Switzerland is high-frequency crypto trading - a new and interesting industry that's captured the interest of many startups.

Usually when people build solutions on AWS, or any other cloud for that matter, they focus on abstraction - load balancers instead of direct access to servers, layered networking, and so on. These are examples of best practices that we employ to build well-architected solutions - secure, low maintenance, and cost-efficient. In cases like crypto trading though, the latency impact of these architectural decisions can represent a poor trade-off, and other options must be considered.
In this article, we will look at the tools and levers that can be used within the AWS platform to drive down latency, both between instances and across the internet.

We will focus on:

- Ensuring our instances have the best chance of low latency *in general*
- Optimizing the latency between the instances we own
- Optimizing the latency from our instances to the rest of the world

The options presented here generally represent a poor trade-off for conventional workloads, and a full understanding of their impact on the security, reliability and cost-effectiveness in particular on your workloads should be arrived at before using any of them. The AWS Well-Architected Framework and Tool are great resources to help you understand the impact of these and other architectural decisions on your workload.

1. Choosing an Instance

Instance Families, Generations, and Sizes

The biggest lever we have in terms of improving network latency is the instance family, generation, and size we choose. As we increase the bandwidth assigned to the interface, we may also see a decrease in latency. These changes have a cost impact, and before committing to using a very large top-tier network instance, we should measure it to see if any latency advantage it yields is worth the additional cost for our workload. At the time of writing the highest throughput available is 100 gigabit, which can be obtained within the following instance families:

- c5n (e.g., c5n.18xlarge, c5n.metal) - compute optimised
- g4dn - GPU instances, useful for machine learning
- inf1 - AWS Inferentia, useful for machine learning inferences
- m5n - balanced CPU and memory
- r5n - memory-optimized

Generally you want the newest instance generation (indicated by the number), an ‘n’ indicating improved networking. The choice between the other details - e.g. memory optimized vs. CPU optimized vs inferentia should be chosen to align with your workload.

The trade-off here means we are inevitably choosing expensive instances to access the highest tier of network performance. Depending on the workload it is likely that these instances will end up underutilized. For typical workloads this would not align with the Well Architected Framework’s Cost Optimization pillar.

Tenancy & Metal Instances

Once we’ve chosen an appropriate instance family and size to match the workload, we move onto the choice of tenancy and metal instances:

	Description	Hypervisor?	Isolation?	Placement Groups?	Pros
Dedicated Host	Reserve a host and explicitly assign VMs to it	Yes	Explicitly assigned VMs	No	Explicitly group VMs onto host to maximize network performance , full control of all neighbors on host
Dedicated Instance	Ensure that VMs only placed on hosts running VMs for the same account	Yes	Shared with other instances within account	Yes	Can use placement groups to mix with regular instances
Metal	Deploy AMIs to bare metal,	No	Single instance only	Yes	Direct access to the underlying hardware

	Description	Hypervisor?	Isolation?	Placement Groups?	Pros
	without a hypervisor				and and hardware feature sets (such as Intel® VT-x) and the ability to run your own hypervisor.
Shared Tenancy	Deploy instances to hosts potentially shared with other accounts	Yes		Yes	Use placement groups to best-effort place instances near each other, spreading instances across hosts lowers impact of any issues arising on underlying hardware.

Choosing a dedicated host allows us to place our instances on the same hardware which is likely to result in the best latency we can expect between the instances. On the other hand, this means that a host failure will bring down all of our instances at once while EC2 moves the VMs onto a new dedicated host. Depending on your workload, this may be an unacceptable failure mode.

Metal instances remove the hypervisor from the equation completely, allowing us to provision standard AMIs to machines without the hypervisor. With EC2's nitro hypervisor and hypervisor cards, the overhead added by the hypervisor is minimal, but nonetheless the impact of this may be a deciding factor for our workload. Metal instances tend to be used mainly where licensing constraints require it, rather than as a performance optimisation.

Ultimately the choice here comes down to the particular workload, how it responds to failures, and how far we want to sacrifice the Well Architected Framework's Reliability pillar to drive our latency numbers down. Once again, the choices made here, focussed on placing the instances as closely together as possible within an AZ, represent a poor trade-off for traditional workloads where reliability is much more important than minimising latency at all costs.

2. Networking - Within the Region

Once we've picked our instances and their tenancy, we might need to hook them up to each other. For low-latency workloads, the less connectivity the better - ultimately avoiding hitting the network is always going to be the fastest option!
In some cases - when the workload can't fit onto a single machine, for instance, or needs connectivity with other, discrete services within your AWS environment - this isn't practical.

Availability Zones

Placing all of our instances into the same AZ will likely give us the best latency between them, but once again, at the expense of reliability. AZs are physically isolated from one another to improve the durability of the region they reside within, and this comes with a small latency cost. The best practice for traditional workloads is to take advantage of services such as EC2 Autoscaling's inbuilt ability to spread instances across availability zones, so that in the unlikely event of an AZ outage, the workload continues to run. Keep in mind that AZs remain close enough for things like RDS' synchronous database replication to work well; it is definitely worth measuring whether or not the improvement here is worth the trade-off!

We must carefully consider whether or not the latency improvements brought by targeting a single AZ only are worth the reliability impact for the workload.

Cluster Placement Groups

Within an AZ, we can go further still and use Cluster Placement Groups to hint to AWS that we want our instances packed closely together on the physical hardware, rather than spread apart - the default behaviour used to improve reliability. It is recommended that all instances for the placement group be launched in a single launch request to increase the likelihood they can be placed effectively.

Network Cards

We need to make sure we are using Enhanced Networking; on the newer instance generations we should have this turned on by default, but it is nonetheless worth checking.

Next, we can look at splitting traffic for different workloads apart onto different interfaces. For instance, for a trading workload, we may choose to separate the interface we use to receive market signals from the interface we use to place orders. This limits the ability of separate streams of traffic to impact each other, and allows us to tune the interfaces separately if required.

Network Cards - Elastic Fabric Adapter

Finally, for traffic within our cluster, we can look at OS bypass networking.

Elastic Fabric Adapter (EFA) is a network card that allows customers to run applications requiring high levels of inter-node communications leveraging operating system bypass. This allows tightly-coupled workloads to achieve improved latency, jitter, and throughput. Using EFAs with their OS bypass functionality imposes some constraints that mean they are not simply a “better ENA”:

- Communications within the same subnet only
- TCP/IP is not available; applications must leverage a library such as Open MPI for communications

EFAs also provide regular, non-kernel-bypass traffic, which can be used to reach the machine regularly, but as discussed earlier, we likely want to separate this class of traffic out onto separate network interfaces anyway.

Network Simplification

Anything in the path of traffic between our instances can affect latency, and we can therefore try to keep these paths as simple as practical while continuing to meet our other architectural goals.

- VPC - use a single VPC where possible; VPC peering, VPN links, and traffic gateway all introduce latency

- Subnets - passing traffic between subnets introduces latency. If we intend to use EFA and kernel-bypass, we will need to keep the clustered portion of our workload inside a single subnet anyway
- Security Groups - attaching multiple SGs, or overly-complex SGs, may impact latency
- Access Control Lists - attaching overly complex ACLs may impact latency

Here too we need to pay special attention to the ramifications of simplification. Simply removing all access control from our network is clearly a step in the wrong direction, particularly in terms of Well Architected’s Security pillar. The problems that we may cause by compromising on security can foreseeably be much more dire than those of our earlier reliability compromises.

Protocol Choice & Serialization

While REST + JSON + HTTP has become the defacto standard of public APIs exposed on the internet, it is much more optimized for common use cases like web applications than it is for low-latency. If we choose to use EFAs and kernel bypass as suggested earlier, we’ve already been pushed away from this for traffic within our cluster, but what if we can’t make this move, or if we have external interfaces?

- If we must use HTTP, favour newer HTTP versions - e.g. HTTP3 / QUIC over HTTP2 over HTTP1
- Favour gRPC over REST
- Favour higher performance serialization options such as protocol buffers over JSON or XML
- Favour websockets PUSH over polling - for instance, if we have incoming market signals

3. Networking - Beyond the Region

While it’s nice to have our instances talking to each other as quickly as possible, most real-world systems, especially trading systems, need to interact with the rest of the world. Fortunately there are some tools we can use here, too, to push our latency down.

PrivateLink & VPC Peering

AWS PrivateLink allows you to publish a service from one VPC into another, without having to directly peer the VPCs and without the traffic travelling over the public internet. In some cases, external API providers provide PrivateLink endpoints for their services in the regions and zones they operate in, which you can then use to

communicate with them.
PrivateLink exposes a Network Load Balancer from one VPC into the other, which may have a latency impact. If you control both accounts you can also use VPC peering or AWS Transit Gateway, which allows traffic to pass between them without the intermediate NLB. This complicates the security situation and couples the two much more tightly together, but may be an appropriate trade-off in some situations.

Global Accelerator & Cloudfront

Global Accelerator allows you to expose a public endpoint for your AWS-hosted service using BGP Anycast such that traffic for the service enters the AWS network as close to the edge as possible. The traffic then travels through AWS' internal network to your service, wherever that may be, rather than across the public internet. This can result in a marked improvement in stability and latency of the connection.

CloudFront is a content delivery network that provides HTTP[S] endpoints to your service distributed around Amazon's global network. Traffic hits CloudFront at the edge, and then travels over AWS' internal network to your service. This can be a simpler alternative to Global Accelerator where HTTP/HTTPS is all that is required.

In our example here - a trading system - it is likely not much use, as we would much rather place our workload as close to the exchange we are using as possible.

Direct Connect Partners

In case you are interested in arbitrage, you may want the lowest latency possible not just between a single region and an exchange, but to multiple exchanges spread out across the world, the idea being to take advantage of price differences quicker than anyone else can. This of course makes the problem much harder.

Some AWS Direct Connect partners such as Equinix operate their own global networks and may offer latency optimized offerings. If you are interested in exploring this possibility you can checkout our AWS' Direct Connect partners.

4. Kernel & Process Optimisation

Once we've gone through all of the "big picture" pieces above, we can move into the instances themselves and start fiddling with local optimisations.

Core Pinning

Core pinning, sometimes also called CPU pinning or processor affinity, allows us to indicate to our OSes scheduler that a particular process should only run on a particular core. This can help avoid cache misses and improve process latency. This sort of optimization warrants benchmarking to ensure that it yields a positive performance impact. Most X86 AWS instance families maintain affinity between virtualized cores and physical hyperthreads, but this can be verified for the relevant family in the documentation. The T2 instance type and graviton instances like T4g are notable exceptions.

TCP & Network Card Tuning

TCP & network card tuning is a rabbit hole that we can also choose to dive into. Ultimately both have a number of levers and tweaks we can play with - window sizes, buffer sizes, and so on - that are generally tuned to provide reliability for normal use cases. Diving into this is beyond the scope of this document!

Conclusions

If you've gotten this far, congratulations! Despite running on virtualized infrastructure, there is still a lot we can do to drive down our latency.

Runner H Viral Content Factory Agent

Check out this winning submission to the Runner H "AI Agent Prompting" Challenge. 👀

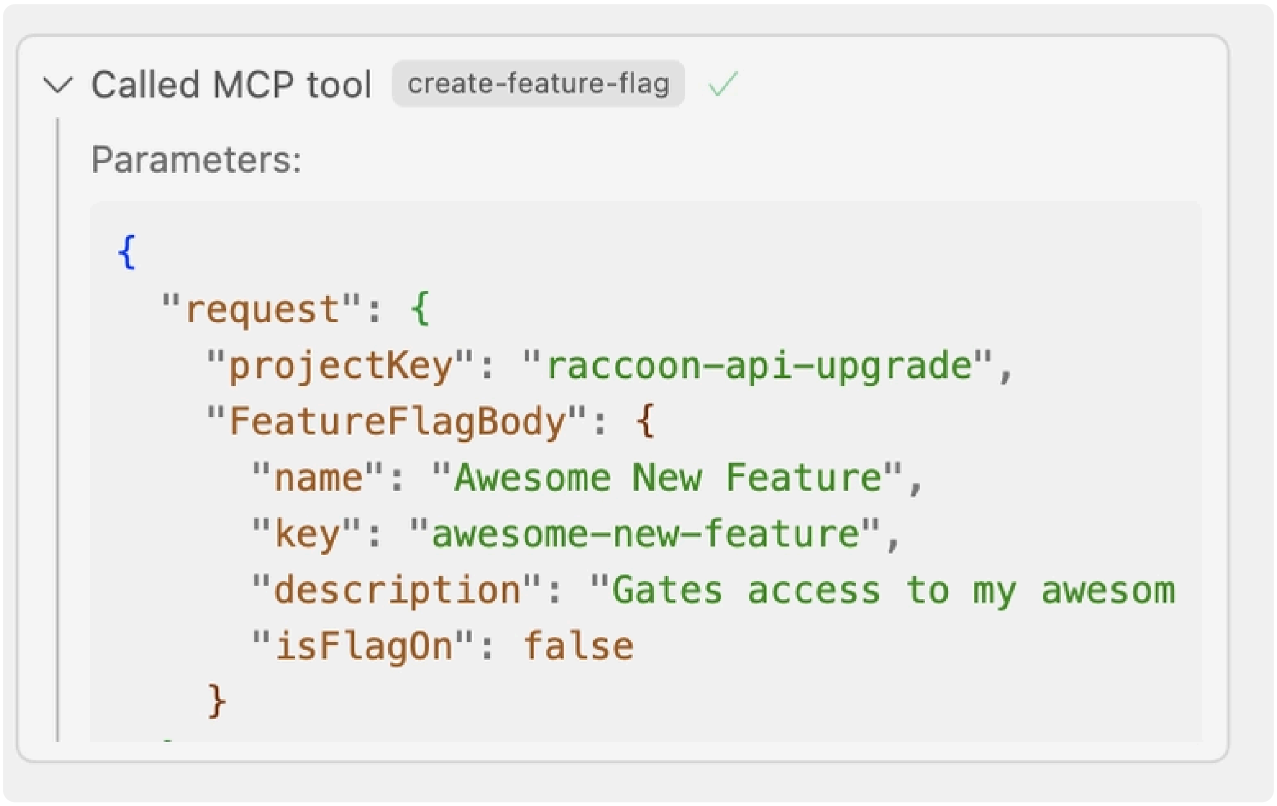
Read more →

Top comments (0)

Code of Conduct • Report abuse

LaunchDarkly PROMOTED

•••



```
{
  "request": {
    "projectKey": "raccoon-api-upgrade",
    "FeatureFlagBody": {
      "name": "Awesome New Feature",
      "key": "awesome-new-feature",
      "description": "Gates access to my awesom
      "isFlagOn": false
    }
```

Create a feature flag in your IDE in 5 minutes with LaunchDarkly's MCP server 🏁

How to create, evaluate, and modify flags from within your IDE or AI client using natural language with LaunchDarkly's new MCP server. Follow along with this tutorial for step by step instructions.

Read full post

AWS Switzerland

More from AWS Switzerland

Join the AWS Community Builders program!
#aws #cloudskills #cloud #switzerland

Anonymize your data using Amazon S3 Object Lambda
#security #aws #cloud #serverless

Monitoring and instrumenting a multi-stage SQS pipeline using Cloudwatch
#aws #cloud #tutorial #cloudwatch

H Company PROMOTED •••

No More Surprises: Get Notified on Terraform Deprecations

Check out this winning submission to the Runner H "AI Agent Prompting" Challenge. 👀

Read more →