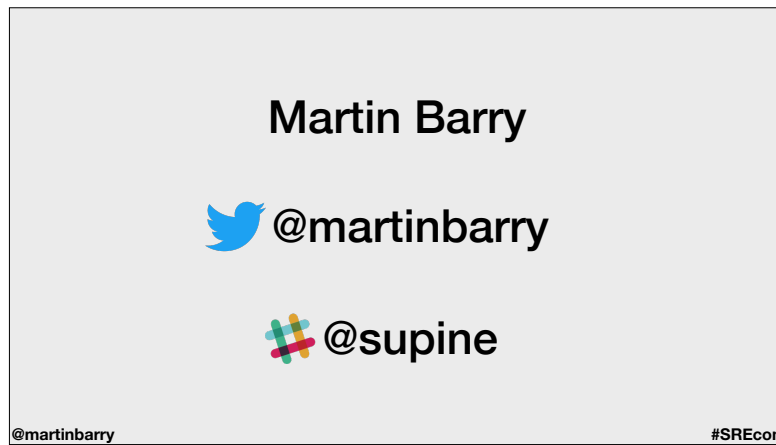


Edge Computing

The Next Frontier for Distributed Systems

Good morning SREcon.

I am here today to talk to you about “Edge Computing”.



I'm Martin, an Australian who moved to Germany 10 years ago.

Those are my Twitter and Slack handles. Feel free to contact me.

I've worked in network and system administration since the year 2000 across a variety of companies and industries.



I currently work for Fastly.

We're an Edge Cloud Platform that helps with high performance delivery of content like websites, APIs and streaming.

You might not have heard of us but our customers are household names in news, e-commerce, video and music.

I want to thank Fastly for funding my travel to be here today. It would not have been possible without their support.

Fastly is hiring. If you want to know more about the company, come to talk to me in the breaks here at the conference or contact me online. I might even have some stickers to hand out.

For the rest of my talk I don't speak for Fastly. These are my observations and opinions.

Define “Edge Computing”

Discuss it’s “features”

@martinbarry

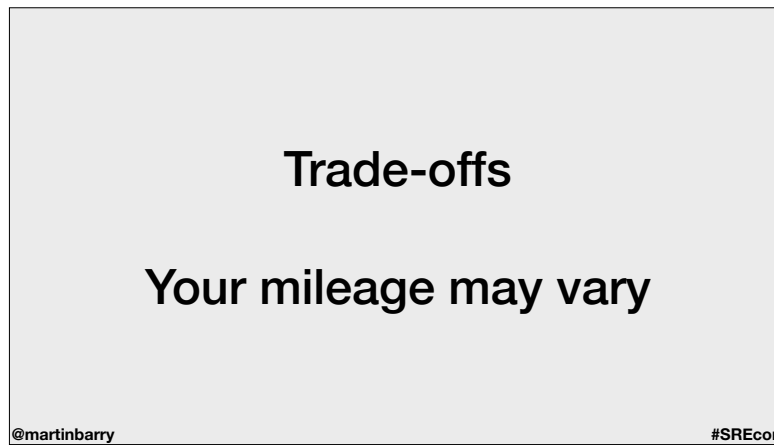
#SREcon

The first goal I have today is to try to define what people might mean when they use the phrase “Edge Computing”.

A bit like the terms “DevOps” and “SRE”, there is no single concept nor implementation that everyone agrees on.

The second goal is an exploration of some of the features of “Edge Computing”, both the good and the bad.

However there are two other themes you’ll see again and again throughout this talk...



The first is “trade-offs”.

Very few of the benefits of “Edge Computing” come for “free”.

Enhancing one aspect of your solution usually means something else gets worse.

The second bonus theme is that everyone’s requirements are different and unique.

What you see as a problem might be someone else’s feature.

Something that you don’t care about can be someone else’s top priority.

Actually, that second statement is not quite right...



...there. Fixed it.

“Edge Computing”

**Executing non-trivial
functionality as close to the
client as is reasonable**

@martinbarry

#SREcon

This is my preferred definition of “Edge Computing”.

People try to define it in different ways but this is the one I keep coming back to.

“Edge Computing”

Executing non-trivial
functionality as close to the
client as is reasonable

@martinbarry

#SREcon

“Executing”

This is not just static assets.

This is not just caching.

We are not simply responding to a request, we are going to manipulate it in some way.

“Edge Computing”

Executing **non-trivial functionality** as close to the client as is reasonable

@martinbarry

#SREcon

“non-trivial functionality”

We should be taking something that we previously thought was only possible at a central server, the origin, and be moving that closer to the client.

“Edge Computing”

**Executing non-trivial
functionality as **close to the
client** as is reasonable**

@martinbarry #SREcon

“close to the client”

We are trying to reduce the network distance between the client and where their request is initially processed.

This is not physical distance. We only care about the logical, topological distance over the network.

But even talking about “distance” is not quite exact.

We only care about distance as a proxy for latency.

Less distance generally equals lower latency network connectivity.

“Edge Computing”

Executing non-trivial
functionality as close to the
client as is **reasonable**

@martinbarry

#SREcon

“reasonable”

At some point trying to get closer and closer to the client no longer makes sense.

The “costs” or “negatives” start to increase at a rate that is too expensive for the “benefits” you achieve.

In the theme of “trade-offs”, “Reasonable” represents balancing the benefits and cost.

In the theme of “your mileage may vary”, the answer to “what is reasonable?” will differ depending on your unique requirements and limitations.

So closer to the client...

...but where exactly?

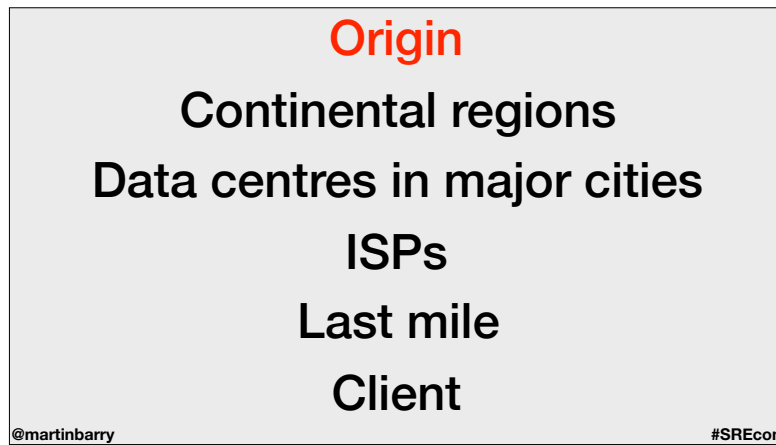
@martinbarry

#SREcon

So I keep talking about being closer to the client but it might help to get a little more specific about where in the Internet edge platforms are built.

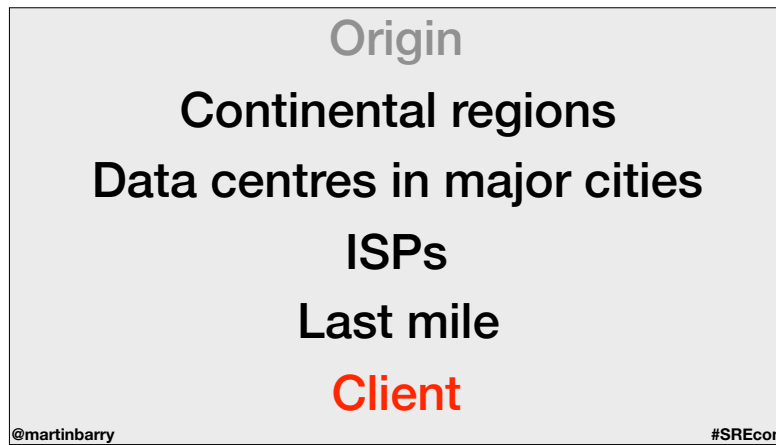


If you start at a single, global origin and traverse all the way down to the client you pass by a number of places where an edge platform can be built.

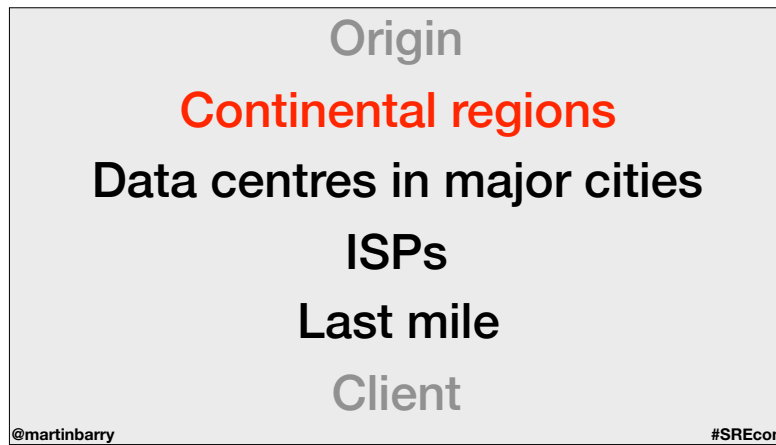


We can exclude the obvious locations.

A single, global origin is the opposite of “edge”.



At the other extreme, running your code client side is a great way to reduce latency but has business and security implications which limits what you can do there.

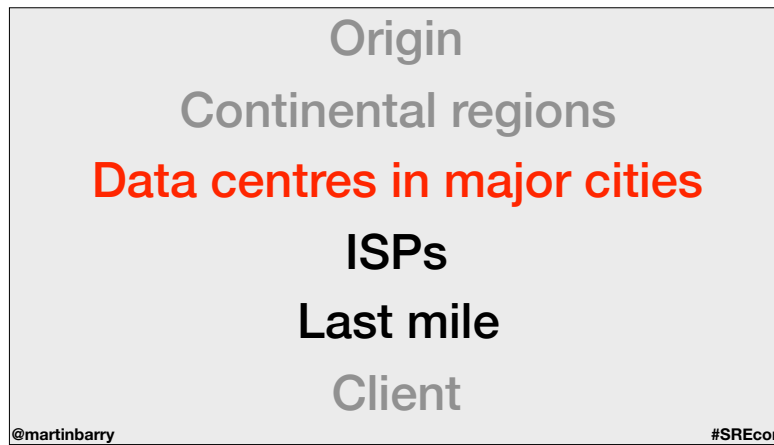


We can also exclude using multiple regions, cloud or otherwise, operated in an active/active fashion.

Although the number of cloud regions is slowly growing, even on the best served continent the largest provider has only half a dozen locations.

Africa and South America tend to be poorly served, if at all.

Large cloud data centres are often near cheap power sources rather than being close to areas of high population density. This results in network distance being higher than you might expect.



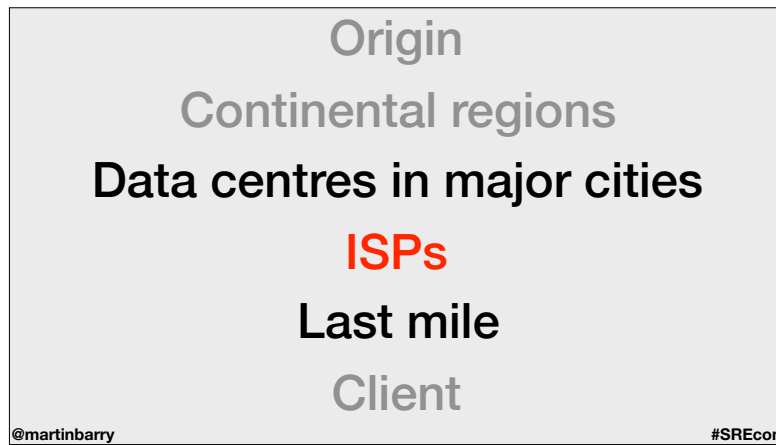
The first layer where you will generally find edge platform servers is in carrier-neutral data centres in major cities.

Carrier-neutral implies that you can connect to many networks and internet exchanges inside the facility.

These tend to be in major cities, particularly ones that have an abundance of optic fibre for networks to use for connectivity.

These facilities and exchanges benefit greatly from network effects. The more networks you can connect to there, the more attractive they are for others to use.

These network effects apply to edge platform providers too. They can serve a lot of different networks from a single, large cluster and achieve very high efficiency.

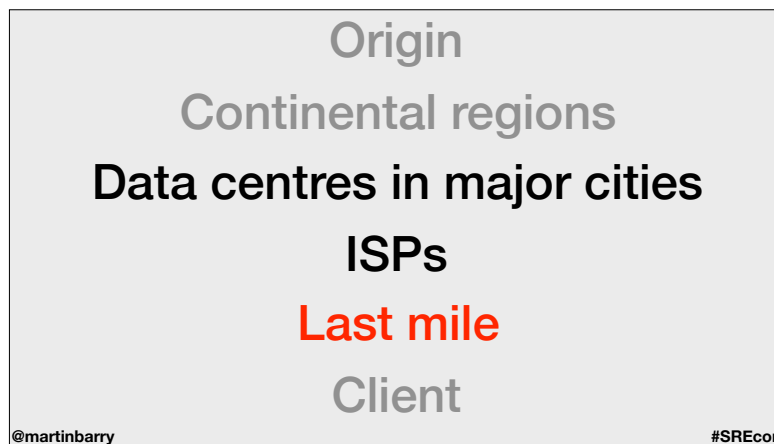


The next layer where you will find edge platform servers is inside an Internet Service Providers network.

In contrast to the carrier-neutral data centres, these clusters can usually only serve the customers of a single ISP.

Often this limitation is a compromise driven by wanting to serve a geographic region where there are limited or no neutral facilities.

Because of the limited pool of clients these clusters would generally be smaller than ones designed to serve multiple networks.



The last layer where you will find edge platform servers is in the so-called “last mile”.

This is the part of the internet that links an internet service providers core network with the homes and offices where their customers live and work. It’s also the part of the network that brings connectivity to your mobile phone.

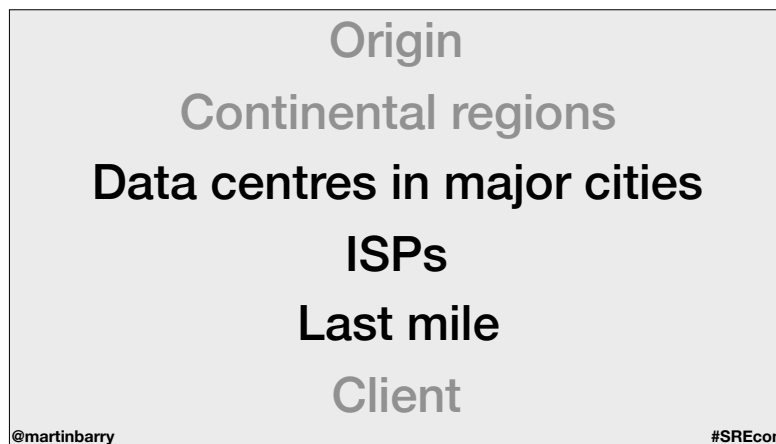
As opposed to the previous two layers, these are not usually located in data centre grade facilities.

That means they often have trade-offs like lower levels of power redundancy and security.

One example might be a telecommunications facility, sometimes referred to as a “central office” or “telephone exchange”.

It might be in a street-side cabinet where a shared physical circuit is demultiplexed down to per-customer circuits, where a transition from fibre to copper or coaxial cable occurs.

Or it might be in the cabinets controlling a mobile phone tower.



So to review...

In broad terms there are three layers where you find edge platform servers.

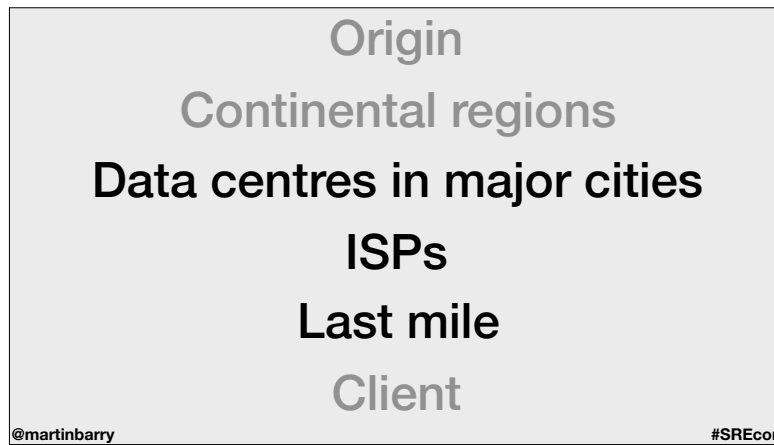
As you move down the layers you get topologically closer to the user.

But you must accept trade-offs to get there.

As the pool of users you can serve gets smaller, the edge provider would generally use smaller clusters. That can be less servers, less powerful servers or both.

You lose the statistical aggregation of a larger cluster. Similar spikes in load have a greater impact on a smaller cluster.

Your requests will be divided amongst more clusters so there will be a higher probability some of them will not see enough requests to stop your processes from being terminated, your cache from being evicted.



You also start to see stranded resources. It does not matter if there is spare capacity in the cluster that serves ISP A if we need more capacity in the cluster that serves ISP B. It does not matter if there is spare capacity in the cluster that serves the mobile network on the south side of town if we need more capacity on the north side of town.

Like a keep repeating, trade-offs. You can chase being closer and closer to the user but the compromises build up.

Will these affect your use of the edge platform? Only you can answer that. Your infrastructure and your user base are unique to your company. Do a proof of concept. Measure all the things. Revisit your assumptions and decisions regularly.

The next request served by a...
different process
different server
different cluster
different provider

@martinbarry

#SREcon

Building web presences at scale requires that we treat every part of the service as stateless as possible. There are few guarantees that the same process will handle two requests from the same client no matter how close together they are made.

This is especially true once you have made the thousands of machines of an edge platform an integrated part of your service.

The next request served by a...

different process

different server

different cluster

different provider

@martinbarry

#SREcon

Even if the next request goes to same cluster, the use of stateless load balancing means that it might be redirected to a different server.

Even if it lands on the same server, high enough load will require multiple processes and the request will be handled by the next available process to avoid delays.

The next request served by a...
different process
different server
different cluster
different provider

@martinbarry

#SREcon

However it's also possible that the next request is received at a totally different cluster.

The simplest explanation for this occurring is the DNS lookup for your hostname expiring and the edge providers traffic management system returning a different cluster IP for the new DNS lookup.

There are other scenarios where this can happen. Sometimes TCP anycast is used to direct traffic to the nearest cluster.

Anycast is when the same IP address or range is announced from multiple locations and the routing systems in between are responsible for where packets land.

Usually there is a single best path to a nearby cluster.

But sometimes there is more than one cluster which are "equidistant", at least from the point of view of the routing system, and an attempt to load balance across the multiple paths lands packets in different clusters.

Each TCP session should reliably go to one cluster, but when a new session is created the different source port will result in the new session hashing differently and possibly going to a different cluster.

The next request served by a...
different process
different server
different cluster
different provider

@martinbarry

#SREcon

Those folks looking to avoid a single point of failure will employ a multiple provider strategy in order to cope with an edge platform outage.

This requires requests be directed to a specific edge platform by your own traffic management system or one provided by a third party.

Again, it's entirely possible that a new DNS lookup is directed differently, either due to load balancing or failure detection.

So two requests from the same client could be handled by two completely different edge platforms.

The next request served by a...
different process
different server
different cluster
different provider

@martinbarry

#SREcon

So to recap.

Scalable solutions require relying on as little state as possible.

This is especially true once you have made the thousands of machines run by your edge provider an integrated part of your service.

You can not predict where the next request from the same client will land.

Request normalisation
Authentication or Paywall
Vary by user agent
Localisation
A / B testing

@martinbarry

#SREcon

We have spoken a lot about “where” so let us move on to some “what”.

Here are a couple of examples...

Request normalisation

Authentication or Paywall

Vary by user agent

Localisation

A / B testing

@martinbarry

#SREcon

Edge computing gives you the power to preprocess requests before they ever reach your origin and other services you might connect out to.

You can sanitise the request to effectively whitelist what reaches your servers.

You can reconstruct the requested path internally to maximise the possibility of the request being a cache hit.

Request parameters often force a cache miss but if they can be added to a synthetic path or ignored after preprocessing you might achieve a cache hit instead.

Request normalisation
Authentication or Paywall
Vary by user agent
Localisation
A / B testing

@martinbarry #SREcon

It's common for a site to display certain content only to logged in users or those with a current subscription.

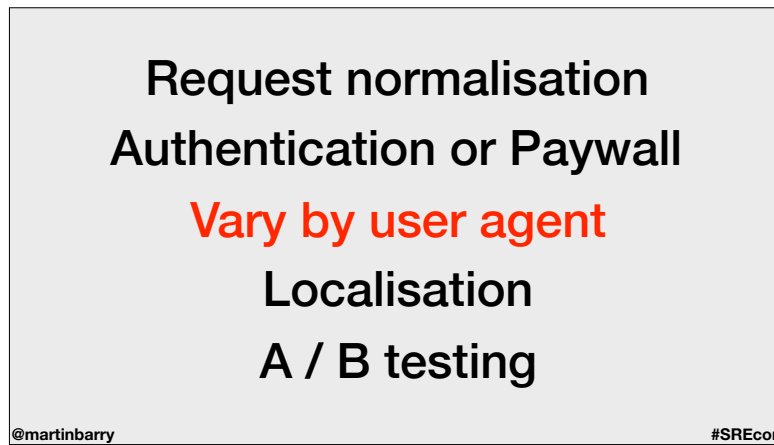
Using edge computing you can send a request to an authorisation service to check the user is logged in or has a valid subscription.

If the check passes you can serve a fully featured version of the page, possibly even from cache.

If the check does not pass, you can serve an alternate page or generate a redirect to your login or membership page.

You might even be able to cache the authentication result so each new request that user makes to the same cluster does not incur that delay.

And if the user logs out or their subscription is no longer valid you can send a purge request to ensure no authorisation remains in any cache.

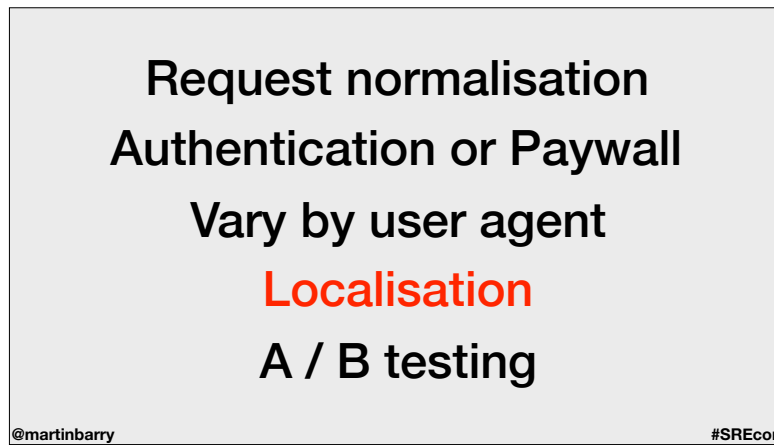


Responsive web design has done a lot for the display of web sites on different screen sizes.

However you can still achieve efficiencies by serving different content to different devices.

Mobile devices can be served smaller images and touch-screen friendly navigation elements.

Rather than redirecting the user to a different subdomain or request path, edge computing allows you to adjust the request internally. This modifies the request to your backend and allows you to cache the different versions but they can still be served under the same URL to the different clients.



Another group of changes you can implement on a per client basis is localisation.

Think of different languages or country specific content to meet different regulations.

You can select defaults for these based on the presence of an accept-language header in the request, which cluster serves the request or any geographic metadata you might have for the client IP address.

If a user selects to override the default language or country selection you can store that as a cookie so subsequent requests respect their selection.

As with the user agent example, you can do this without needing to redirect the client. This enables the same external URL to flexibly serve the localised content, perhaps even from cache.

Request normalisation
Authentication or Paywall
Vary by user agent
Localisation
A / B testing

@martinbarry

#SREcon

As products and services are developed it's common to use A / B testing to allow some users to try an alternative version so you can measure the impact or collect feedback.

Edge computing allows you to implement this as close to the user as possible.

Your code running on the edge can select which bucket a new user will be placed in and then use a cookie to deliver a consistent experience to them.

Again, it can all be done internally so you can cache and serve the alternatives without the user seeing a change in URL.

Domain specific languages (DSLs)
Generic programming languages
Containers

@martinbarry

#SREcon

We are now getting into the “how” of edge computing.

The implementations differ in how much of the runtime is controlled by you and how much is controlled by the provider.

Domain specific languages (DSLs)
Generic programming languages
Containers

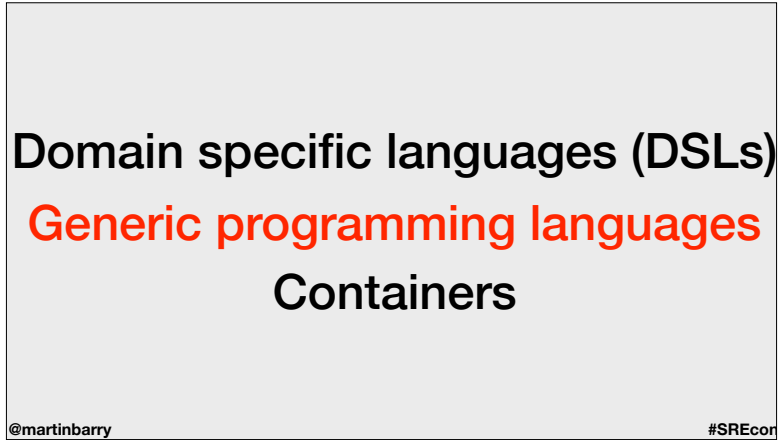
@martinbarry

#SREcon

Domain specific languages, or DSLs, are specialised languages designed to achieve a specific task or are tied to specific software.

If you are familiar with configuration management systems, a good example of a DSL is the one used by Puppet. It is customised to suit the mostly declarative nature of configuration management.

Edge DSLs are adapted to being very good at manipulating requests with a highly efficient runtime and tight integration with the cache.



Domain specific languages (DSLs)
Generic programming languages
Containers

@martinbarry

#SREcon

The next type of implementation allows you to use a generic programming language.

This offers your developers the possibility of familiarity with a language that might already be in use in other parts of your service.

While it's possible to ship non-standard libraries and extend the capabilities of your code running at the edge, keep in mind the resource limitations it will run under.

You will have a limited memory footprint which needs to cover all of your code, the non-standard libraries and any workspace you need for buffering and manipulation of data.

You will also have a time limit to process each request. Usually this only counts "CPU time" which is the cumulative time of when your code is actually running.

Domain specific languages (DSLs)
Generic programming languages
Containers

@martinbarry

#SREcon

Finally some providers let you ship a full container and will run it on their edge platform.

This gives you the most control as you are responsible for most of the runtime.

But it also limits the efficiencies the provider can extract. Each container contains all the binaries and libraries it requires, resulting in a lot of duplicate, redundant copies in memory and on disk.

WebAssembly (Wasm)

**WebAssembly System Interface
(WASI)**

@martinbarry

#SREcon

Before I finish talking about code I just want to quickly mention two technologies which are starting to reshape how edge computing is implemented.

WebAssembly (Wasm)

WebAssembly System Interface (WASI)

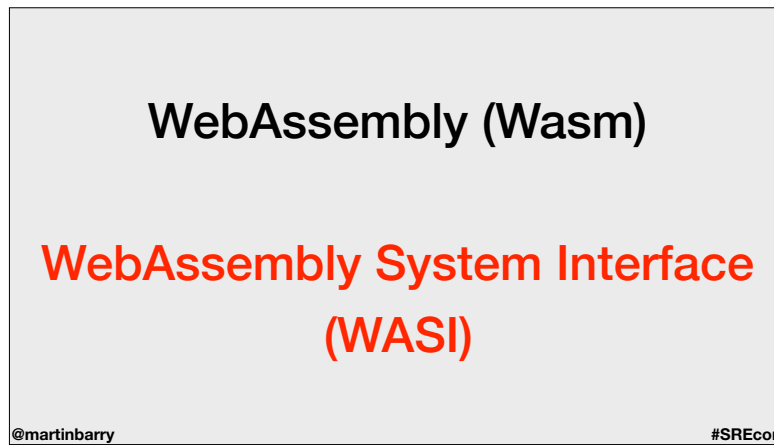
@martinbarry

#SREcon

WebAssembly, or Wasm for short, is a binary instruction format originally designed to run sandboxed inside web browsers.

By compiling code written in higher level languages down to WebAssembly bytecode it can be run safely and at near native speeds.

Firefox, Chrome, Safari and Edge all support it.



What makes WebAssembly exciting for edge platforms is having those safety and performance benefits outside a web browser.

WebAssembly System Interface, or WASI for short, is a project started by Mozilla which aims to define a standard for WebAssembly to run at the operating system level.

There are a couple of implementations of the WASI standard in development and they are showing promising signs of high speed and resource efficiency.

Coldstart times in microseconds instead of milliseconds.

Per module overhead of kilobytes instead of megabytes.

WebAssembly definitely looks like it will play a major part in how edge computing functions in the future.



The last major feature of edge computing I'd like to cover is access to data at the edge.

Edge providers usually provide some kind of simple key-value database.

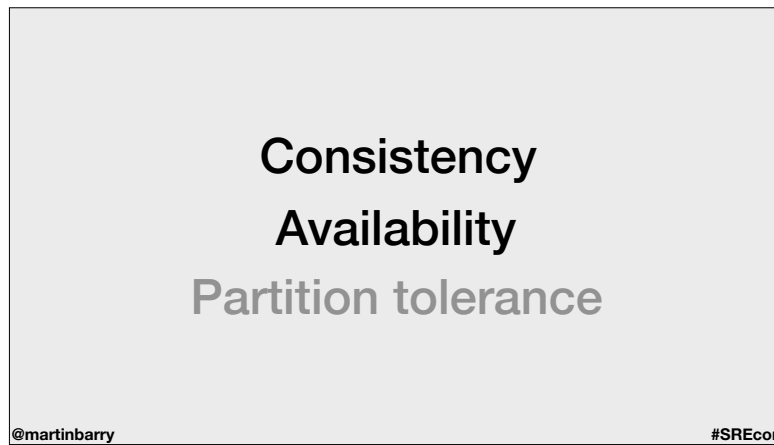
But they vary in how they are implemented and where they fit into the CAP trade-off.

Consistency
Availability
Partition tolerance

@martinbarry

#SREcon

For those not familiar with CAP it stands for “consistency”, “availability” and “partition tolerance”.



Your database must be resistant to network partitions so you can only choose some compromise between consistency and availability.

Consistency requires that every read receives the most recent write. If we can not guarantee that, an error should be returned instead.

Availability requires we respond to each read even if the result we return might be stale.

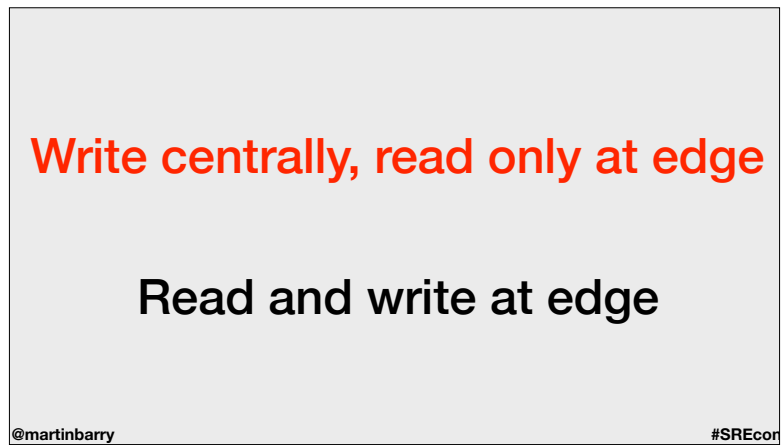
Write centrally, read only at edge

Read and write at edge

@martinbarry

#SREcon

Getting back to data at the edge, in my mind this kind of service falls into two broad categories...



The first type is a centrally updated database which then pushes the dataset out to all edge machines.

Code running at the edge can read from it but not write to it.

This fits in perfectly with slowly changing, read heavy data.

The central database will be at the consistency end of the CAP spectrum.

However the edge will be closer to the availability end. Updates will be pushed out and the edge datasets will converge on the most recent version of the data. You might hear this model referred to as “eventual consistency”.

Edge reads will not block even when updates are still being pushed or even not being received.

There will be a limit to how stale the edge provider will let data become. If there is sufficient disruption a machine or cluster will often be removed from service to prevent ongoing impact to your service.

Write centrally, read only at edge

Read and write at edge

@martinbarry

#SREcon

The second type is both read from and written to at the edge.

This provides a simple way for your edge solution to both use and modify the database.

However being able to write at the cluster or regional level means that there will be consistency compromises.

You will need to carefully research and test how the provider arbitrates between clashing writes from different locations and consider how this impacts your use of this type of database.

Write centrally, read only at edge

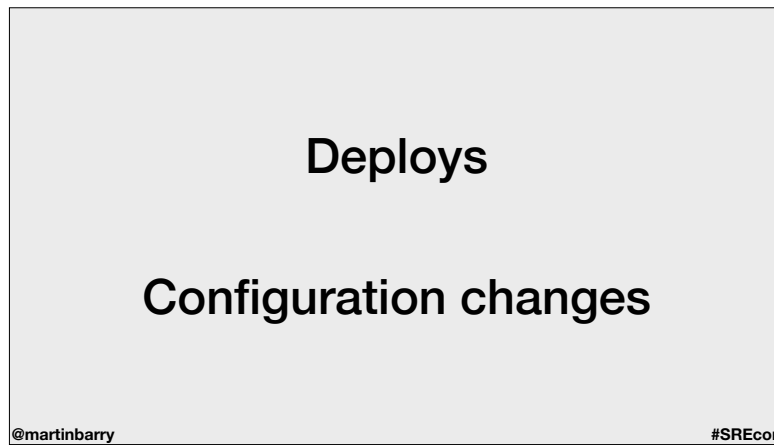
Read and write at edge

@martinbarry

#SREcon

To recap, access to data greatly enhances the capabilities of edge computing.

However the highly distributed nature of an edge platform means that it will always work best with read heavy use cases that tolerate eventual consistency.



In my last few minutes I'd like to cover some of the important operational aspects of edge computing.

Once you make an edge platform part of your service you now have to include it in your workflows for code deploys and configuration changes.

Some, perhaps even all, of these updates should be possible via API so you can integrate it with your continuous integration and delivery solution.

Updates now need to be pushed out to thousands of machines spread all over the world.

You should research and test how long it takes for an edge provider to complete that push.

This delay has important implications for how long you will have multiple versions of your code or configuration running and how long it will take to rollback a bad update.

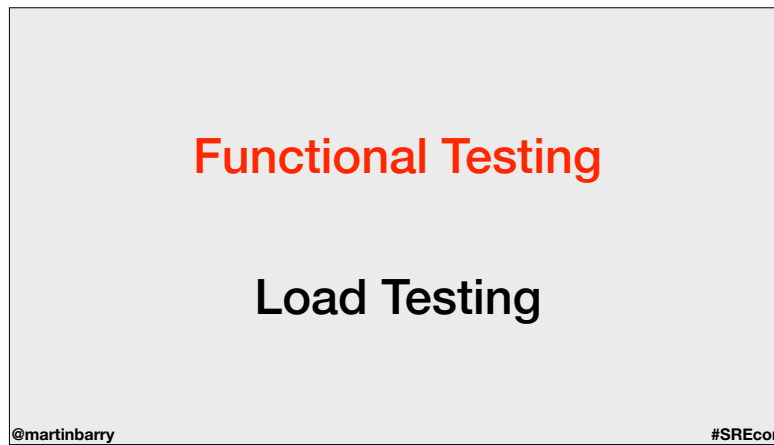
Functional Testing

Load Testing

@martinbarry

#SREcon

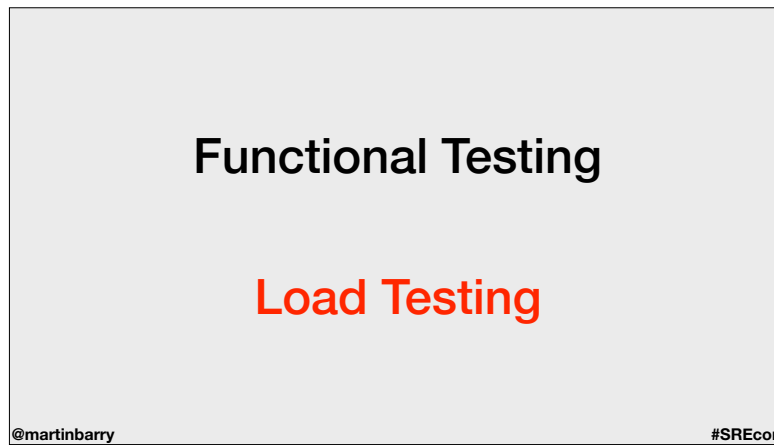
Once you have code and configuration deploying you will want to run tests against it.



Functional testing is mostly simple, your test cases exercising the different parts of your edge application.

What can be tricky is when your code relies on something like the geographic metadata of the client IP, in which case you will need a special handler in your code to allow your tests to override those variables.

If your code relies on which cluster is handling the request to alter behaviour or perhaps use a different backend, you might need to determine how to run tests against specific clusters so you can exercise a representative sample of them.



Load testing is particularly tricky with edge computing.

Edge platforms are specifically designed to be very good at serving a highly distributed set of clients.

Any concentrated testing infrastructure located in data centres or the cloud can not replicate the kind of load generated by organic traffic.

The best method of load testing an edge platform is with live traffic.

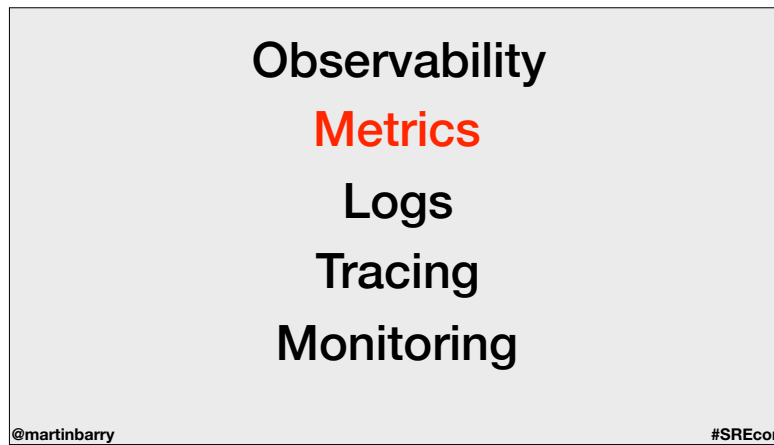
While testing in production generally has a bad reputation, it's the only reliable way to see how your code and the platform work together under load.

**Observability
Metrics
Logs
Tracing
Monitoring**

@martinbarry

#SREcon

Once you have integrated an edge platform into your service it now must be part of your observability considerations.



Most edge platforms will offer some basic metrics via a dashboard in their management portal.

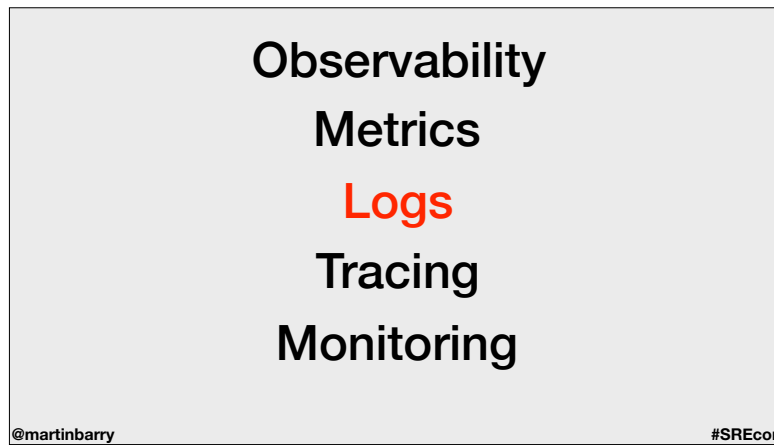
They may also make those metrics and more available to you via an API so you can pull them into your own metrics system.

There may also be the possibility of emitting metrics yourself from each request.

However you should keep in mind that doing so keeps the process on that request and delays it being available to handle another request.

You could consider batching the metrics from multiple requests handled by the same process but you run the risk of the process being prematurely terminated before it can do so leading to a large loss of data-points.

There is another alternative which we'll come back to after talking about logging...



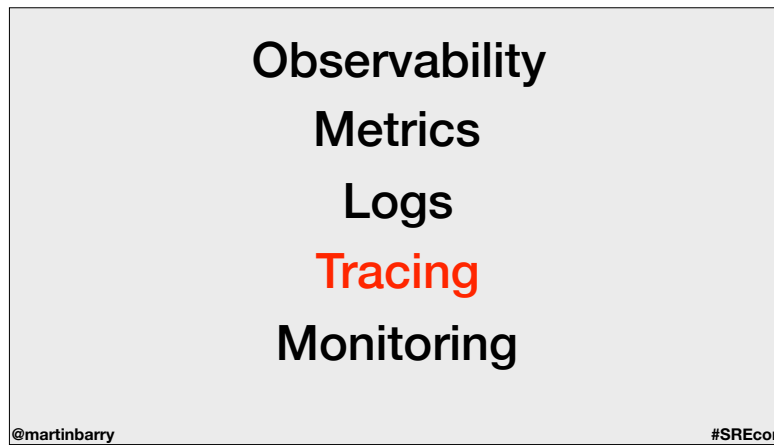
Most edge platforms offer you the logs generated by every request handled at the edge.

Logs can be delivered to you in various ways. Some examples include streaming them to your logging service, writing them directly to your cloud storage or offering them up for you to download.

You should check if the default logging format meets your requirements and what customisation is possible if you need something different.

You should also determine how much delay there is between the request occurring and you having access to the logging of the event. Obviously the smaller the delay, the more useful the logs will be for insight and troubleshooting.

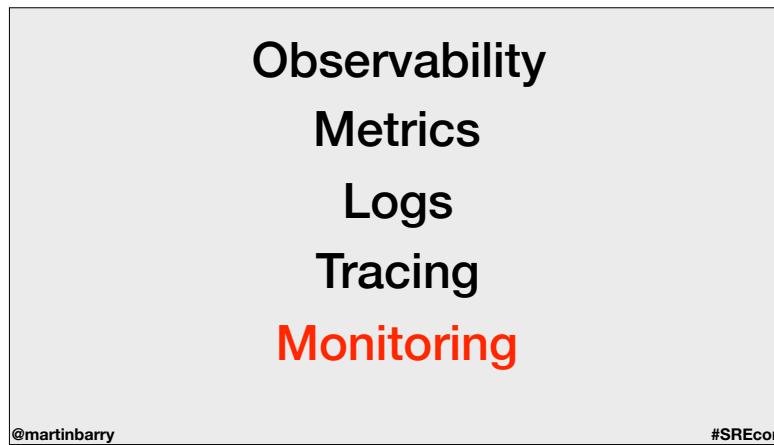
I mentioned earlier another way of getting metrics. Your logs offer a source of data-points from which you can derive metrics. This is often the most flexible and reliable way to get custom metrics out of an edge platform.



Tracing is the ability to link together the different parts of a system which handled the components of a request.

An example might be a request that is received at the edge, a call is made to an authentication service which succeeds, then a request is made to the origin because the content was not in cache. To fully inspect and troubleshoot a request like that you need a unique request ID to be propagated to each part.

When you start using an edge platform you should start generating the unique request IDs on the edge. If you do not, you've made it much harder to include the edge spans in your tracing tool and reliably link them to the other parts of the request.



When an edge platform is a critical part of your service you will want to do some monitoring of your own to be confident that the platform's reporting of problems or outages is timely, consistent and transparent.

You can use one of the many external monitoring services to poll either important URLs of your service or a specific health-check one.

However they can't detect all issues that impact your users as the testing locations tend to be in well connected data centres and cloud providers.

Some kind of client-side "real user monitoring" might be necessary to truly capture the experience of your users.

Thank you!

supine.com/srecon19apac

 **@martinbarry**

 **@supine**

@martinbarry

#SREcon

...and that's all I have time for today. You can find my slides at that URL.

I hope I've been able to explain a little about what "edge computing" actually is, what you can do with it and how you might think about operating with it integrated into your service.

Thanks for listening and feel free to chat to me in the hallways or contact me online.