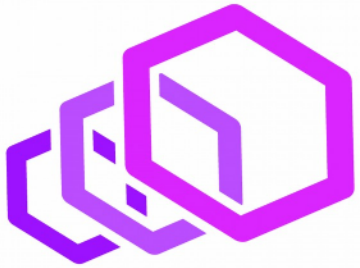
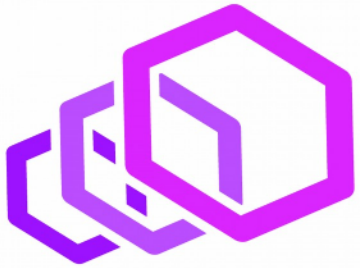


envoycon

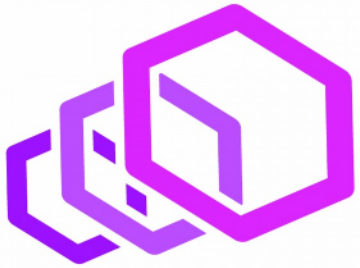
Cynthia Coan
@TheTrueKungfury
https://github.com/SecurityInsanity
cynthia@coan.dev



envoycon

Making Envoy Sustainable

Cynthia Coan
@TheTrueKungfury
https://github.com/SecurityInsanity
cynthia@coan.dev



envoycon

~~Making Envoy Sustainable~~

How to be an responsible platform engineering team and have your service last.

Cynthia Coan
@TheTrueKungfury
https://github.com/SecurityInsanity
cynthia@coan.dev

Setting the Scene

Setting the Scene

"Alright we'll give you time to build your API Gateway/Service Mesh/<buzzword of the week>."

... And since you've been so talkative about it, we'll let you help shape the project, and be on the team.

Setting the Scene

"Alright we'll give you time to build your API Gateway/Service Mesh/<buzzword of the week>."

... And since you've been so talkative about it, we'll let you help shape the project, and be on the team.

You're imaginary self has never done this before...

Sustainability...

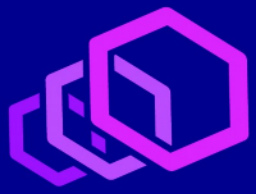
So You Get To Thinking, and After Thinking About It...

Sustainability...

So You Get To Thinking, and After Thinking About It...

You want to ***keep technical debt low*** since techincal debt impacts multiple teams.

Sustainability...

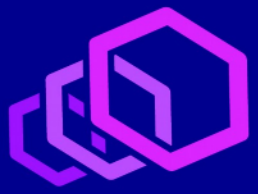


envoycon

So You Get To Thinking, and After Thinking About It...

You want to ***keep technical debt low*** since techincal debt impacts multiple teams.
You want to ensure ***envoy is updated frequently*** to keep delivering value.

Sustainability...

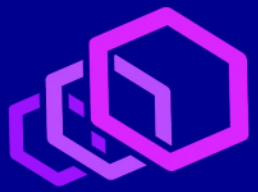


envoycon

So You Get To Thinking, and After Thinking About It...

You want to ***keep technical debt low*** since techincal debt impacts multiple teams.
You want to ensure ***envoy is updated frequently*** to keep delivering value.
You want to be able to make it ***easy to do the right thing***, and hard to do the wrong thing.

Sustainability...



envoycon

So You Get To Thinking, and After Thinking About It...

You want to ***keep technical debt low*** since techincal debt impacts multiple teams.
You want to ensure ***envoy is updated frequently*** to keep delivering value.
You want to be able to make it ***easy to do the right thing***, and hard to do the wrong thing.

You want to keep up with Envoy.

So You Start Writing Software...

So you start writing software, get a team to work with you, and start rolling out software...

So You Start Writing Software...

So you start writing software, get a team to work with you, and start rolling out software...

So why don't people roll out software?

So You Start Writing Software...

So you start writing software, get a team to work with you, and start rolling out software...

So why don't people roll out software?

- ***Time***

So You Start Writing Software...

So you start writing software, get a team to work with you, and start rolling out software...

So why don't people roll out software?

- ***Time***
- ***Poorly Defined Breaking Change Policies***

So Why Is Time A Problem?

So you do some time digging with teams, and you realize something...

So Why Is Time A Problem?

So you do some time digging with teams, and you realize something...

Teams have a lot going on. Like a Lot.

So Why Is Time A Problem?

So you do some time digging with teams, and you realize something...

Teams have a lot going on. Like a Lot.

They have ***App Updates***.

So Why Is Time A Problem?

So you do some time digging with teams, and you realize something...

Teams have a lot going on. Like a Lot.

They have ***App Updates***.
They have ***Infrastructure to Maintain***.

So Why Is Time A Problem?

So you do some time digging with teams, and you realize something...

Teams have a lot going on. Like a Lot.

They have ***App Updates***.
They have ***Infrastructure to Maintain***.
They have ***Support Issues***.

So Why Is Time A Problem?

So you do some time digging with teams, and you realize something...

Teams have a lot going on. Like a Lot.

They have ***App Updates***.
They have ***Infrastructure to Maintain***.
They have ***Support Issues***.
They have ***Feature Work***.

So Why Is Time A Problem?

So you do some time digging with teams, and you realize something...

Teams have a lot going on. Like a Lot.

They have ***App Updates***.
They have ***Infrastructure to Maintain***.
They have ***Support Issues***.
They have ***Feature Work***.

Remember it's not cause they're unwilling. They just have little time.

Breaking Changes...

Breaking Changes...

So is this slide just going to be "Use SemVer?"

Breaking Changes...

So is this slide just going to be "Use SemVer?"

Nah. In fact it may be worse with SemVer.

Breaking Changes...

So is this slide just going to be "Use SemVer?"

Nah. In fact it may be worse with SemVer.

It's cause SemVer got one thing wrong.

Breaking Changes...

So is this slide just going to be "Use SemVer?"

Nah. In fact it may be worse with SemVer.

It's cause SemVer got one thing wrong.

```cpp
#include <vector>

class A {};
class B {};

class C {
 public:
  explicit C() {}

  void filter(std::vector<A> vec) {}
};

class D : public A, public B {
 public:
  explicit D() {}
};

int main() {
  (new C())->filter({D(), D(), D()});
  return 0;
}
```

Breaking Changes...

So is this slide just going to be "Use SemVer?"

Nah. In fact it may be worse with SemVer.

It's cause SemVer got one thing wrong.

```cpp
#include <vector>

class A {};
class B {};

class C {
 public:
  explicit C() {}

  void filter(std::vector<A> vec) {}
  // This has caused a build break:
  // `error: call to member function 'filter'
  // is ambiguous`
  void filter(std::vector<B> vec) {}
};

class D : public A, public B {
 public:
  explicit D() {}
};

int main() {
  (new C())->filter({D(), D(), D()});
  return 0;
}
```

Breaking Changes...

So is this slide just going to be "Use SemVer?"

Nah. In fact it may be worse with SemVer.

It's cause SemVer got one thing wrong.

```
#include <string_view>
#include <gtest/gtest.h>

TEST(AStaticFunction, LogOutput) {
  World world = new World(new Civilization());
  LOGGER_RECORD_OUTPUT();
  world->do_large_thing();
  const auto log_output =
        LOGGER_RECORDING_STOP_AND_GET_LOGS();
  EXPECT_EQ(
        log_output[0],
    std::string_view("[File.cpp:42] success")
  );
}
```

Breaking Changes...

So is this slide just going to be "Use SemVer?"

Nah. In fact it may be worse with SemVer.

It's cause SemVer got one thing wrong.

```
1  class A {
2   public:
3    explicit A();
4
5    [[deprecated("use b() for better perf")]]
6          void a();
7
8     void b();
9  };
10
11 int main() {
12   return 0;
13 }
```

Breaking Changes...

So is this slide just going to be "Use SemVer?"

Nah. In fact it may be worse with SemVer.

It's cause SemVer got one thing wrong.

Context Matters.

```
1  class A {
2   public:
3    explicit A();
4
5    [[deprecated("use b() for better perf")]]
6          void a();
7
8     void b();
9  };
10
11 int main() {
12   return 0;
13 }
```

A Solution

So how do we fix this as a team? What can you do as a platform team?

A Solution

So how do we fix this as a team? What can you do as a platform team?

You make your upgrades easy.

A Solution

So how do we fix this as a team? What can you do as a platform team?

You make your upgrades easy.

Like Really Easy.

For Your Codebases...

So what's the secret, for codebases you own?

For Your Codebases...

So what's the secret, for codebases you own?

Communicate wither other teams.

For Your Codebases...

So what's the secret, for codebases you own?

Communicate wither other teams.

Document ***more than a public API***, and test everything.

For Your Codebases...

So what's the secret, for codebases you own?

Communicate wither other teams.

Document ***more than a public API***, and test everything.

Invest seriously to ***make sure your API is followed***.

For Your Codebases...

So what's the secret, for codebases you own?

Communicate wither other teams.

Document ***more than a public API***, and test everything.

Invest seriously to ***make sure your API is followed***.

When was the last time you...

For Your Codebases...

So what's the secret, for codebases you own?

Communicate wither other teams.

Document ***more than a public API***, and test everything.

Invest seriously to ***make sure your API is followed***.

When was the last time you...

wrote a script to do an atomic code change, and let tests validate it?

For Your Codebases...

So what's the secret, for codebases you own?

Communicate wither other teams.

Document ***more than a public API***, and test everything.

Invest seriously to ***make sure your API is followed***.

When was the last time you...

wrote a script to do an atomic code change, and let tests validate it?

wrote a custom lint rule to check for usage of unstable apis?

wrote a custom lint rule to check for anti-patterns?

For Code You Don't Control...

So if you don't control code, and can't make it easy for your intern. What can you do to make it so an intern can do it?

For Code You Don't Control...

So if you don't control code, and can't make it easy for your intern. What can you do to make it so an intern can do it?

Give Them Time.

For Code You Don't Control...

So if you don't control code, and can't make it easy for your intern. What can you do to make it so an intern can do it?

Give Them Time.

If you start looking at a release the day it's released, or when they cut an RC. It's too late.

For Code You Don't Control...

So if you don't control code, and can't make it easy for your intern. What can you do to make it so an intern can do it?

Give Them Time.

If you start looking at a release the day it's released, or when they cut an RC. It's too late.

How can you test earlier than an RC Though?

Pro's & Cons

Pro's & Cons

- When upgrades are consistently easy, ***people don’t mind doing your upgrades*** soon.
- Requires more work from the people providing the service for multiple teams.

Pro's & Cons

- When upgrades are consistently easy, ***people don't mind doing your upgrades*** soon.
- When upgrades are easy, even if the team is under a crunch period, ***they still have time for your upgrades***.

- Requires more work from the people providing the service for multiple teams.
- Needs to be constantly reviewed.

Pro's & Cons

- When upgrades are consistently easy, ***people don't mind doing your upgrades*** soon.
- When upgrades are easy, even if the team is under a crunch period, ***they still have time for your upgrades***.
- By more accurately defining what the purpose of your project is, and helping others understand how your code should/should not be used. You get less: "***why are you doing it this way***?"

- Requires more work from the people providing the service for multiple teams.
- Needs to be constantly reviewed.
- Requires Talking with Coworkers, which may be harder for teams on different timezones.

Another Bump In The Road

So you keep building, and you think tech debt is under control. Using the same process as normal. However, you're in pain, why?

Another Bump In The Road

So you keep building, and you think tech debt is under control. Using the same process as normal. However, you're in pain, why?

Remember what I said at the beginning. ***Tech Debt affects more than you***.

Another Bump In The Road

So you keep building, and you think tech debt is under control. Using the same process as normal. However, you're in pain, why?

Remember what I said at the beginning. ***Tech Debt affects more than you***.

What about your teams image? ***Will people want to keep using your product***?

Another Bump In The Road

So you keep building, and you think tech debt is under control. Using the same process as normal. However, you're in pain, why?

Remember what I said at the beginning. ***Tech Debt affects more than you***.

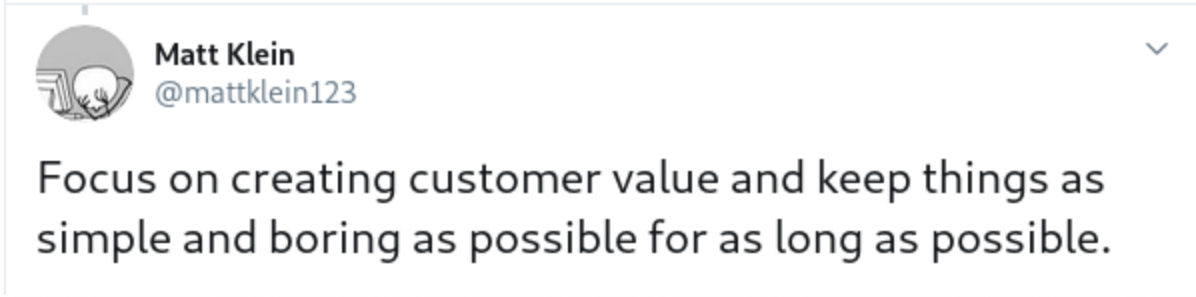
What about your teams image? ***Will people want to keep using your product***?

At the same time, you can't be scared to make changes. ***Tech Debt will always happen***.

When Planning...

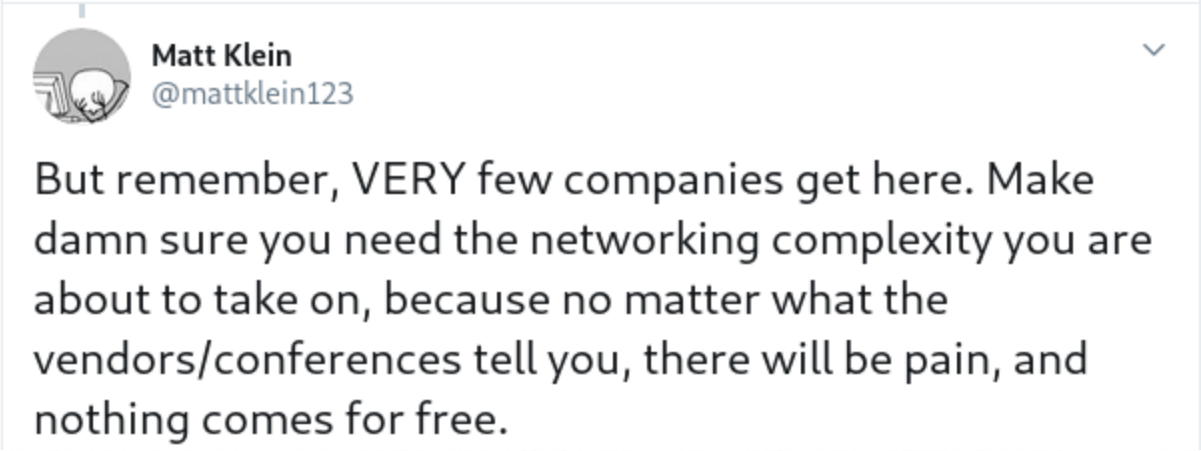
When Planning...

- ***"Keep it Boring"***



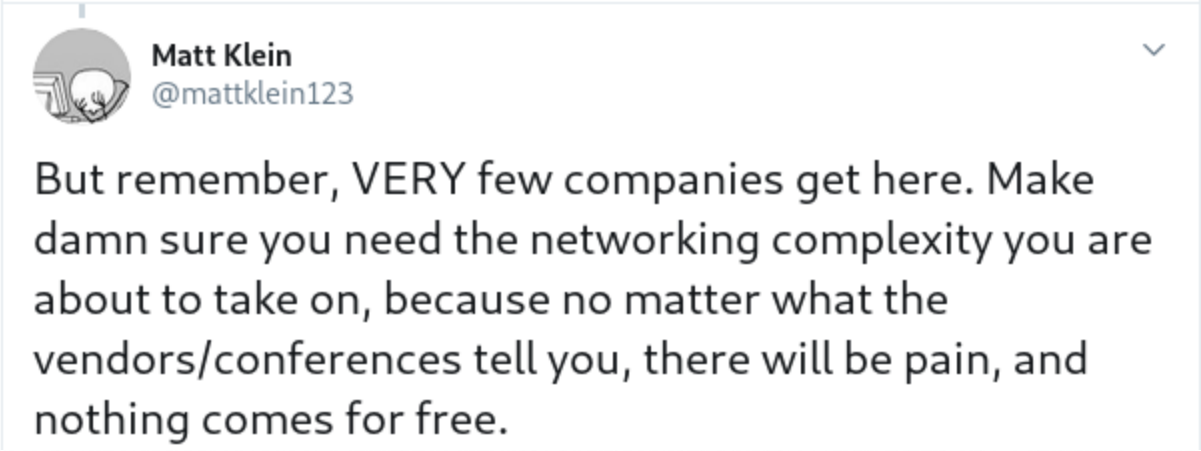
When Planning...

- ***"Keep it Boring"***
- Remember every bit of tech adds complexity.



When Planning...

- ***"Keep it Boring"***
- Remember every bit of tech adds complexity.
  - ***A Managed Solution isn't inherently better.***

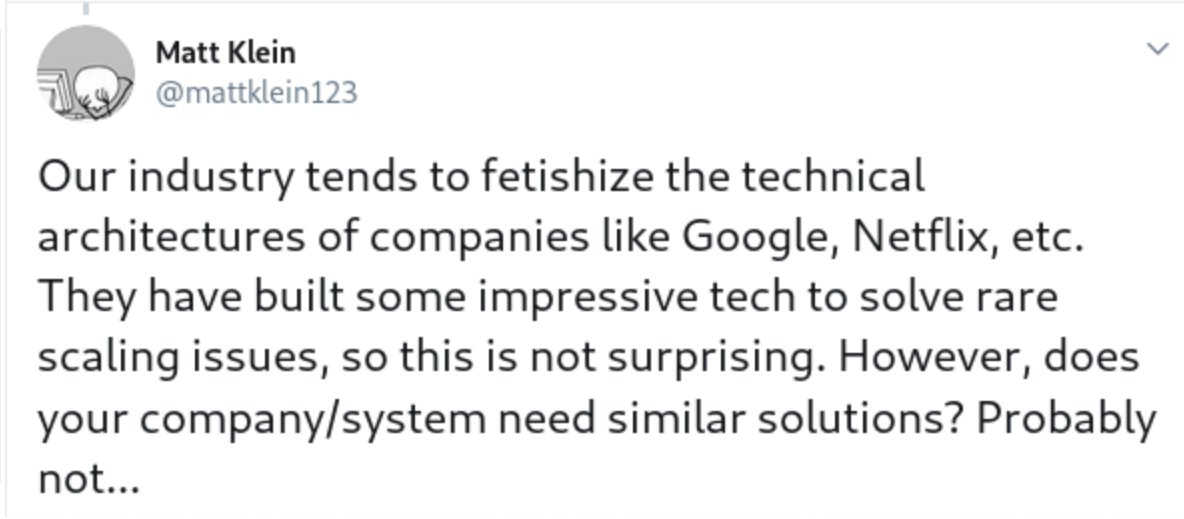


When Planning...

- ***"Keep it Boring"***
- Remember every bit of tech adds complexity.
  - ***A Managed Solution isn't inherently better.***
- Do you really need all the features of that open-source filter?

When Planning...

- ***"Keep it Boring"***
- Remember every bit of tech adds complexity.
  - ***A Managed Solution isn't inherently better.***
- Do you really need all the features of that open-source filter?
  - Or can you lock some of it down with a linter, ***so it's not adopted accidentally until you're ready to handle it***?



Pros & Cons

Pros & Cons

- By planning features ahead of time you can help with the previous problem of versioning.

- Planning Features can sometimes take too long to be a source of severe blockage, and is easy to screw up.

Pros & Cons

- By planning features ahead of time you can help with the previous problem of versioning.
- Less Tech Debt gives you more time to work on feature work, and solving new problems which is always interesting.

- Planning Features can sometimes take too long to be a source of severe blockage, and is easy to screw up.
- We all hate being in meetings all day, which is possible if you don't limit the service review/syncs you need.

Pros & Cons

- By planning features ahead of time you can help with the previous problem of versioning.
- Less Tech Debt gives you more time to work on feature work, and solving new problems which is always interesting.
- People will actually want to contribute to your codebase because they don't have to understand the tech debt that exist in the codebase.

- Planning Features can sometimes take too long to be a source of severe blockage, and is easy to screw up.
- We all hate being in meetings all day, which is possible if you don't limit the service review/syncs you need.
- Prioritizing is hard work, since some things really are small but really important to someone.

An Example

An Example

Every new feature requires a tech plan, test plan, and agreements.

An Example

Every new feature requires a tech plan, test plan, and agreements.

We have a linter that powers automated code-reviews, and enforces best practices.

An Example

Every new feature requires a tech plan, test plan, and agreements.

We have a linter that powers automated code-reviews, and enforces best practices.

Constantly developed on master, CI always uses master, and nightly tests.

An Example

Every new feature requires a tech plan, test plan, and agreements.

We have a linter that powers automated code-reviews, and enforces best practices.

Constantly developed on master, CI always uses master, and nightly tests.

Keep Meeting Heavy Days to one day of the week so planning around it is easier.

An Example

Every new feature requires a tech plan, test plan, and agreements.

We have a linter that powers automated code-reviews, and enforces best practices.

Constantly developed on master, CI always uses master, and nightly tests.

Keep Meeting Heavy Days to one day of the week so planning around it is easier.

Deprecations in config are tracked separately and fixed asap.

An Example

Every new feature requires a tech plan, test plan, and agreements.

We have a linter that powers automated code-reviews, and enforces best practices.

Constantly developed on master, CI always uses master, and nightly tests.

Keep Meeting Heavy Days to one day of the week so planning around it is easier.

Deprecations in config are tracked separately and fixed asap.

Made the decision to not use xDS for more than secrets.

An Example

Every new feature requires a tech plan, test plan, and agreements.

We have a linter that powers automated code-reviews, and enforces best practices.

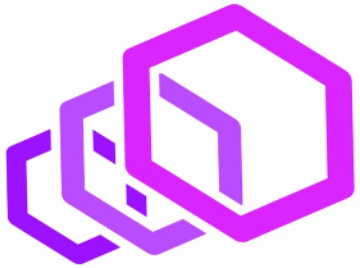
Constantly developed on master, CI always uses master, and nightly tests.

Keep Meeting Heavy Days to one day of the week so planning around it is easier.

Deprecations in config are tracked separately and fixed asap.

Made the decision to not use xDS for more than secrets.

Invented a special versioning scheme to better communicate guarantees, and what is expected from maintainers, and consumers.



envoycon

Thank You!

Cynthia Coan
@TheTrueKungfury
https://github.com/SecurityInsanity
cynthia@coan.dev