DevOps Anti-patterns

Or, when the best is enemy of the good (enough)

Michele Finelli
m@biodec.com
BioDec

Some background

DevOps has some *mantras*:

- Culture
- Automation
- Monitoring
- Sharing

Some background

DevOps has some *mantras*:

- Culture
- Automation
- Monitoring
- Sharing

Some background

DevOps has some *mantras*:

- Culture
- Automation
- Monitoring
- Sharing

Some background

DevOps has some *mantras*:

- Culture
- Automation
- Monitoring
- Sharing

Some background

DevOps has some *mantras*:

- Culture
- Automation
- Monitoring
- Sharing

And some consequences

Like:

- Monitor all the things
- Infrastructure as code
- Continuous integration, continuous deployment

And much more, depending on your industry, technology, hype of the moment, etcetera.

And some consequences

Like:

- Monitor all the things
- Infrastructure as code
- Continuous integration, continuous deployment

And much more, depending on your industry, technology, hype of the moment, etcetera.

And some consequences

Like:

- Monitor all the things
- Infrastructure as code
- Continuous integration, continuous deployment

And much more, depending on your industry, technology, hype of the moment, etcetera.

And some consequences

Like:

- Monitor all the things
- Infrastructure as code
- Continuous integration, continuous deployment

And much more, depending on your industry, technology, hype of the moment, etcetera.

And some consequences

Like:

- Monitor all the things
- Infrastructure as code
- Continuous integration, continuous deployment

And much more, depending on your industry, technology, hype of the moment, etcetera.

The great DevOps failure ?

That was the title of another talk that I proposed, but it was down voted, so let's continue with this one.

Seriously: I do not think that DevOps is a failure, *au contraire*, only that, like drugs, it has to be taken with care, and perhaps even under medical supervision.

The great DevOps failure ?

That was the title of another talk that I proposed, but it was down voted, so let's continue with this one.
Seriously: I do not think that DevOps is a failure, *au contraire*, only that, like drugs, it has to be taken with care, and perhaps even under medical supervision.

DevOps anti-patterns

- I will show some misuses of the DevOps principles, things that I have seen happening in companies I worked for.
- Or mistakes I did by myself.
- Sometimes it was an excess of zeal.
- Sometimes it was a misunderstanding of the principles.

DevOps anti-patterns

- I will show some misuses of the DevOps principles, things that I have seen happening in companies I worked for.
- Or mistakes I did by myself.
- Sometimes it was an excess of zeal.
- Sometimes it was a misunderstanding of the principles.

DevOps anti-patterns

- I will show some misuses of the DevOps principles, things that I have seen happening in companies I worked for.
- Or mistakes I did by myself.
- Sometimes it was an excess of zeal.
- Sometimes it was a misunderstanding of the principles.

DevOps anti-patterns

- I will show some misuses of the DevOps principles, things that I have seen happening in companies I worked for.
- Or mistakes I did by myself.
- Sometimes it was an excess of zeal.
- Sometimes it was a misunderstanding of the principles.

DevOps anti-patterns

- I will show some misuses of the DevOps principles, things that I have seen happening in companies I worked for.
- Or mistakes I did by myself.
- Sometimes it was an excess of zeal.
- Sometimes it was a misunderstanding of the principles.

Index

Back to basics

So, which are the objectives of "Infrastructure as code" ?

- Repeatable.
- Testable.
- Speed.
- . . . name your own.

Back to basics

So, which are the objectives of "Infrastructure as code" ?

- Repeatable.
- Testable.
- Speed.
- . . . name your own.

Back to basics

So, which are the objectives of "Infrastructure as code" ?

- Repeatable.
- Testable.
- Speed.
- . . . name your own.

Back to basics

So, which are the objectives of "Infrastructure as code" ?

- Repeatable.
- Testable.
- Speed.
- . . . name your own.

Back to basics

So, which are the objectives of "Infrastructure as code" ?

- Repeatable.
- Testable.
- Speed.
- . . . name your own.

Risk analysis

- The advantages of automation are obvious, but the effort to build and **maintain** the code is often overlooked.
- As with everything, the right thing to do is to weight the advantages and their costs again the **risks** of alternatives.
- Alternatives. *Noun, plural*, as in “There are more alternatives”.
- Not automating everything is very different from no automation at all. Very.

Risk analysis

- The advantages of automation are obvious, but the effort to build and **maintain** the code is often overlooked.
- As with everything, the right thing to do is to weight the advantages and their costs again the **risks** of alternatives.
- Alternatives. *Noun, plural*, as in “There are more alternatives”.
- Not automating everything is very different from no automation at all. Very.

Risk analysis

- The advantages of automation are obvious, but the effort to build and **maintain** the code is often overlooked.
- As with everything, the right thing to do is to weight the advantages and their costs again the **risks** of alternatives.
- Alternatives. *Noun, plural*, as in “There are more alternatives”.
- Not automating everything is very different from no automation at all. Very.

Risk analysis

- The advantages of automation are obvious, but the effort to build and **maintain** the code is often overlooked.
- As with everything, the right thing to do is to weight the advantages and their costs again the **risks** of alternatives.
- Alternatives. *Noun, plural*, as in “There are more alternatives”.
- Not automating everything is very different from no automation at all. Very.

Risk analysis

- The advantages of automation are obvious, but the effort to build and **maintain** the code is often overlooked.
- As with everything, the right thing to do is to weight the advantages and their costs again the **risks** of alternatives.
- Alternatives. *Noun, plural*, as in “There are more alternatives”.
- Not automating everything is very different from no automation at all. Very.

Suppose that . . .

- . . . there is a system that you **know** (with a very good degree of accuracy) that is **not going to change** for quite a long time (years).
- . . . you have something that has really **low value**, something that, in case, can stay broken for days, even weeks, before somebody notices.
- . . . there is a service that is so awkward, or so **hostile to humans** that to automate it you need an herculean effort (and you have no demi-god around).

Suppose that . . .

- . . . there is a system that you **know** (with a very good degree of accuracy) that is **not going to change** for quite a long time (years).
- . . . you have something that has really **low value**, something that, in case, can stay broken for days, even weeks, before somebody notices.
- . . . there is a service that is so awkward, or so **hostile to humans** that to automate it you need an herculean effort (and you have no demi-god around).

Suppose that . . .

- . . . there is a system that you **know** (with a very good degree of accuracy) that is **not going to change** for quite a long time (years).
- . . . you have something that has really **low value**, something that, in case, can stay broken for days, even weeks, before somebody notices.
- . . . there is a service that is so awkward, or so **hostile to humans** that to automate it you need an herculean effort (and you have no demi-god around).

Suppose that . . .

- . . . there is a system that you **know** (with a very good degree of accuracy) that is **not going to change** for quite a long time (years).
- . . . you have something that has really **low value**, something that, in case, can stay broken for days, even weeks, before somebody notices.
- . . . there is a service that is so awkward, or so **hostile to humans** that to automate it you need an herculean effort (and you have no demi-god around).

. . . then automation may not be the best for you!

- Facing the risk of having to rebuild the system, or reconfigure the service, or else could be **cheaper** than building and maintaining the infrastructure code.
- Not automating something does not mean **cowboy programming** or sloppy procedures.
- It means that there has been an analysis of the alternatives among automation and **various degrees** of manual intervention.

. . . then automation may not be the best for you!

- Facing the risk of having to rebuild the system, or reconfigure the service, or else could be **cheaper** than building and maintaining the infrastructure code.
- Not automating something does not mean **cowboy programming** or sloppy procedures.
- It means that there has been an analysis of the alternatives among automation and **various degrees** of manual intervention.

. . . then automation may not be the best for you!

- Facing the risk of having to rebuild the system, or reconfigure the service, or else could be **cheaper** than building and maintaining the infrastructure code.
- Not automating something does not mean **cowboy programming** or sloppy procedures.
- It means that there has been an analysis of the alternatives among automation and **various degrees** of manual intervention.

. . . then automation may not be the best for you!

- Facing the risk of having to rebuild the system, or reconfigure the service, or else could be **cheaper** than building and maintaining the infrastructure code.
- Not automating something does not mean **cowboy programming** or sloppy procedures.
- It means that there has been an analysis of the alternatives among automation and **various degrees** of manual intervention.

Do your homework

- Leave an **updated documentation** of the procedure that had to be automated.
- Create a **checklist** to test that the procedure has been applied correctly.
- Know how to **ask for help** (*i.e.* do not let the knowledge disappear from your company).

Do your homework

- Leave an **updated documentation** of the procedure that had to be automated.
- Create a **checklist** to test that the procedure has been applied correctly.
- Know how to **ask for help** (*i.e.* do not let the knowledge disappear from your company).

Do your homework

- Leave an **updated documentation** of the procedure that had to be automated.
- Create a **checklist** to test that the procedure has been applied correctly.
- Know how to **ask for help** (*i.e.* do not let the knowledge disappear from your company).

Do your homework

- Leave an **updated documentation** of the procedure that had to be automated.
- Create a **checklist** to test that the procedure has been applied correctly.
- Know how to **ask for help** (*i.e.* do not let the knowledge disappear from your company).

Index

An hammer is not a screwdriver

- The landscape of alternative ways to build a given application is often overwhelming. People need to simplify (and cannot learn and know everything, obviously).
- So, on one hand, trying to leverage existing expertise is a smart thing to do.
- On the other hand, sometimes a tool is simply *wrong* for a given task.

An hammer is not a screwdriver

- The landscape of alternative ways to build a given application is often overwhelming. People need to simplify (and cannot learn and know everything, obviously).
- So, on one hand, trying to leverage existing expertise is a smart thing to do.
- On the other hand, sometimes a tool is simply *wrong* for a given task.

An hammer is not a screwdriver

- The landscape of alternative ways to build a given application is often overwhelming. People need to simplify (and cannot learn and know everything, obviously).
- So, on one hand, trying to leverage existing expertise is a smart thing to do.
- On the other hand, sometimes a tool is simply *wrong* for a given task.

An hammer is not a screwdriver

- The landscape of alternative ways to build a given application is often overwhelming. People need to simplify (and cannot learn and know everything, obviously).
- So, on one hand, trying to leverage existing expertise is a smart thing to do.
- On the other hand, sometimes a tool is simply *wrong* for a given task.

Initial state *versus* final state

- There could be confusion between tools built to create an **initial state with given properties** and tools whose task is to **maintain a given state**.
- An example of the former is **provisioning a virtual machine**.
- An example of the latter is **ensuring a given security profile** — you may think to a list of `iptables` rules, or an Apparmor or Selinux profile.

Initial state *versus* final state

- There could be confusion between tools built to create an **initial state with given properties** and tools whose task is to **maintain a given state**.
- An example of the former is **provisioning a virtual machine**.
- An example of the latter is **ensuring a given security profile** — you may think to a list of `iptables` rules, or an Apparmor or Selinux profile.

Initial state *versus* final state

- There could be confusion between tools built to create an **initial state with given properties** and tools whose task is to **maintain a given state**.
- An example of the former is **provisioning a virtual machine**.
- An example of the latter is **ensuring a given security profile** — you may think to a list of `iptables` rules, or an Apparmor or Selinux profile.

Initial state *versus* final state

- There could be confusion between tools built to create an **initial state with given properties** and tools whose task is to **maintain a given state**.
- An example of the former is **provisioning a virtual machine**.
- An example of the latter is **ensuring a given security profile** — you may think to a list of `iptables` rules, or an Apparmor or Selinux profile.

Initial state, or *fire and forget*

- Initial state tools are similar to `bash` scripts.
- They do something and then, if you change something, **they do not care** at all (well, they do not even have the capabilities to notice it).
- They can be tweaked to do **long and complex things**, if needed.

Initial state, or *fire and forget*

- Initial state tools are similar to `bash` scripts.
- They do something and then, if you change something, **they do not care** at all (well, they do not even have the capabilities to notice it).
- They can be tweaked to do **long and complex things**, if needed.

Initial state, or *fire and forget*

- Initial state tools are similar to `bash` scripts.
- They do something and then, if you change something, **they do not care** at all (well, they do not even have the capabilities to notice it).
- They can be tweaked to do **long and complex things**, if needed.

Initial state, or *fire and forget*

- Initial state tools are similar to `bash` scripts.
- They do something and then, if you change something, **they do not care** at all (well, they do not even have the capabilities to notice it).
- They can be tweaked to do **long and complex things**, if needed.

Final state, or *being possessed by a daemon*

- Final state tools are similar to `cron` jobs.
- They *happen* at a certain time and do something. Again. And again. And again and again and again, ***ad infinitum***.
- If you mess with their business, they notice, and **they will react** in don Vito Corleone's style.
- Better they do **simple tasks**, so that two consecutive scripts do not compete with each other . . . or you get a glimpse of the meaning of the word *pandemonium*.

Final state, or *being possessed by a daemon*

- Final state tools are similar to `cron` jobs.
- They *happen* at a certain time and do something. Again. And again. And again and again and again, ***ad infinitum***.
- If you mess with their business, they notice, and **they will react** in don Vito Corleone's style.
- Better they do **simple tasks**, so that two consecutive scripts do not compete with each other . . . or you get a glimpse of the meaning of the word *pandemonium*.

Final state, or *being possessed by a daemon*

- Final state tools are similar to `cron` jobs.
- They *happen* at a certain time and do something. Again. And again. And again and again and again, ***ad infinitum***.
- If you mess with their business, they notice, and **they will react** in don Vito Corleone's style.
- Better they do **simple tasks**, so that two consecutive scripts do not compete with each other . . . or you get a glimpse of the meaning of the word *pandemonium*.

Final state, or *being possessed by a daemon*

- Final state tools are similar to `cron` jobs.
- They *happen* at a certain time and do something. Again. And again. And again and again and again, ***ad infinitum***.
- If you mess with their business, they notice, and **they will react** in don Vito Corleone's style.
- Better they do **simple tasks**, so that two consecutive scripts do not compete with each other . . . or you get a glimpse of the meaning of the word *pandemonium*.

Final state, or *being possessed by a daemon*

- Final state tools are similar to `cron` jobs.
- They *happen* at a certain time and do something. Again. And again. And again and again and again, ***ad infinitum***.
- If you mess with their business, they notice, and **they will react** in don Vito Corleone's style.
- Better they do **simple tasks**, so that two consecutive scripts do not compete with each other . . . or you get a glimpse of the meaning of the word *pandemonium*.

Know your business

- Problem is, tools creators add features to better serve their users and in good faith they risk changing a tool perfectly fitted for one job into something sub-optimal for many jobs (to be honest, this more a disease of the closed, proprietary software world).
- Noticing a bed smell and switching tools can save you tons of time later, when you will fight them to do stuff they were not meant to do.
- All of your systems will be obsolete, eventually: so do not fear to move to newer and better alternatives.

Know your business

- Problem is, tools creators add features to better serve their users and in good faith they risk changing a tool perfectly fitted for one job into something sub-optimal for many jobs (to be honest, this more a disease of the closed, proprietary software world).
- Noticing a bed smell and switching tools can save you tons of time later, when you will fight them to do stuff they were not meant to do.
- All of your systems will be obsolete, eventually: so do not fear to move to newer and better alternatives.

Know your business

- Problem is, tools creators add features to better serve their users and in good faith they risk changing a tool perfectly fitted for one job into something sub-optimal for many jobs (to be honest, this more a disease of the closed, proprietary software world).
- Noticing a bed smell and switching tools can save you tons of time later, when you will fight them to do stuff they were not meant to do.
- All of your systems will be obsolete, eventually: so do not fear to move to newer and better alternatives.

Know your business

- Problem is, tools creators add features to better serve their users and in good faith they risk changing a tool perfectly fitted for one job into something sub-optimal for many jobs (to be honest, this more a disease of the closed, proprietary software world).
- Noticing a bed smell and switching tools can save you tons of time later, when you will fight them to do stuff they were not meant to do.
- All of your systems will be obsolete, eventually: so do not fear to move to newer and better alternatives.

Index

An easy piece

- Ok, imagine that you had a system that was really important, and there was **real value** automating all the procedures involving it . . .
- . . . also, picture yourself in choosing exactly the **right set of tools**, and then writing the automation code . . .
- . . . only to realize that now you have to maintain a very **thin layer** of a couple of Chef/Puppet/Ansible/Whenever lines of code wrapped around . . .

An easy piece

- Ok, imagine that you had a system that was really important, and there was **real value** automating all the procedures involving it . . .
- . . . also, picture yourself in choosing exactly the **right set of tools**, and then writing the automation code . . .
- . . . only to realize that now you have to maintain a very **thin layer** of a couple of Chef/Puppet/Ansible/Whenever lines of code wrapped around . . .

An easy piece

- Ok, imagine that you had a system that was really important, and there was **real value** automating all the procedures involving it . . .
- . . . also, picture yourself in choosing exactly the **right set of tools,** and then writing the automation code . . .
- . . . only to realize that now you have to maintain a very **thin layer** of a couple of Chef/Puppet/Ansible/Whenever lines of code wrapped around . . .

An easy piece

- Ok, imagine that you had a system that was really important, and there was **real value** automating all the procedures involving it . . .
- . . . also, picture yourself in choosing exactly the **right set of tools,** and then writing the automation code . . .
- . . . only to realize that now you have to maintain a very **thin layer** of a couple of Chef/Puppet/Ansible/Whenever lines of code wrapped around . . .

Surprise !

TWO THOUSAND LINES OF BASH* CODE !

* or Powershell, if that is your sexual orientation.

Surprise !

TWO THOUSAND LINES OF BASH* CODE !

* or Powershell, if that is your sexual orientation.

Refactoring

- Infrastructure as code means that you have to deal with . . . code ! It seems a silly remark, but in practice automation code, once working, is seldom refactored.
- Automation code is not a **second class** citizen of your environment.
- Apply to infrastructure code all the (good) practices that you apply to the software that you develop.

Refactoring

- Infrastructure as code means that you have to deal with . . . code ! It seems a silly remark, but in practice automation code, once working, is seldom refactored.
- Automation code is not a **second class** citizen of your environment.
- Apply to infrastructure code all the (good) practices that you apply to the software that you develop.

Refactoring

- Infrastructure as code means that you have to deal with . . . code ! It seems a silly remark, but in practice automation code, once working, is seldom refactored.
- Automation code is not a **second class** citizen of your environment.
- Apply to infrastructure code all the (good) practices that you apply to the software that you develop.

Refactoring

- Infrastructure as code means that you have to deal with . . . code ! It seems a silly remark, but in practice automation code, once working, is seldom refactored.
- Automation code is not a **second class** citizen of your environment.
- Apply to infrastructure code all the (good) practices that you apply to the software that you develop.

Index

What's monitoring ?

- If you do not have already read it, go for chapter 16 of Cesarini and Vinoski's "Designing for Scalability with Erlang/OTP".
- They do a great job in describing that monitoring is a mix of metrics management, event logs management and alerting for failures.
- *Properly* monitoring a system is more complex of what it looks. Often **much more** complex.

What's monitoring ?

- If you do not have already read it, go for chapter 16 of Cesarini and Vinoski's "Designing for Scalability with Erlang/OTP".
- They do a great job in describing that monitoring is a mix of metrics management, event logs management and alerting for failures.
- *Properly* monitoring a system is more complex of what it looks. Often **much more** complex.

What's monitoring ?

- If you do not have already read it, go for chapter 16 of Cesarini and Vinoski's "Designing for Scalability with Erlang/OTP".
- They do a great job in describing that monitoring is a mix of metrics management, event logs management and alerting for failures.
- *Properly* monitoring a system is more complex of what it looks. Often **much more** complex.

What's monitoring ?

- If you do not have already read it, go for chapter 16 of Cesarini and Vinoski's "Designing for Scalability with Erlang/OTP".
- They do a great job in describing that monitoring is a mix of metrics management, event logs management and alerting for failures.
- *Properly* monitoring a system is more complex of what it looks. Often **much more** complex.

Keeping track of things is hard

- Monitoring all aspects of all your systems is probably what you do **not** want.
- Keep track of what you really, really need. Really. Add as many extra “really” to convince you.
- Each little alert, every tiny metrics, every extra line in your log is **extra work**.

Keeping track of things is hard

- Monitoring all aspects of all your systems is probably what you do **not** want.
- Keep track of what you really, really need. Really. Add as many extra “really” to convince you.
- Each little alert, every tiny metrics, every extra line in your log is **extra work**.

Keeping track of things is hard

- Monitoring all aspects of all your systems is probably what you do **not** want.
- Keep track of what you really, really need. Really. Add as many extra “really” to convince you.
- Each little alert, every tiny metrics, every extra line in your log is **extra work**.

Keeping track of things is hard

- Monitoring all aspects of all your systems is probably what you do **not** want.
- Keep track of what you really, really need. Really. Add as many extra “really” to convince you.
- Each little alert, every tiny metrics, every extra line in your log is **extra work**.

Maintenance

- Monitoring is related to **maintenance**: after all if you are not going to react to an alert, why are you receiving it in the first place ? Simply send them to Dave Null.
- So, please take notice: each bit of extra work (even just one more email to read) adds to the pile that you already have **for ever and ever**.

Maintenance

- Monitoring is related to **maintenance**: after all if you are not going to react to an alert, why are you receiving it in the first place ? Simply send them to Dave Null.
- So, please take notice: each bit of extra work (even just one more email to read) adds to the pile that you already have **for ever and ever**.

Maintenance

- Monitoring is related to **maintenance**: after all if you are not going to react to an alert, why are you receiving it in the first place ? Simply send them to Dave Null.
- So, please take notice: each bit of extra work (even just one more email to read) adds to the pile that you already have **for ever and ever**.

Hygiene

I have some empirical rules to keep the environment healthy (and my sanity too).

- If you **add** some monitoring, then you to **remove** some monitoring (from somewhere else).
- If you cannot remove anything, it is time to either add **people** or dismiss clients.
- Technology may reduce the effort to do some monitoring, but that is often a **tricky gain** — did you reduce your burden after you introduced a time-series database in your infrastructure, or just won some more work to do ? Just asking for a friend . . .

Hygiene

I have some empirical rules to keep the environment healthy (and my sanity too).

- If you **add** some monitoring, then you to **remove** some monitoring (from somewhere else).
- If you cannot remove anything, it is time to either add **people** or dismiss clients.
- Technology may reduce the effort to do some monitoring, but that is often a **tricky gain** — did you reduce your burden after you introduced a time-series database in your infrastructure, or just won some more work to do ? Just asking for a friend . . .

Hygiene

I have some empirical rules to keep the environment healthy (and my sanity too).

- If you **add** some monitoring, then you to **remove** some monitoring (from somewhere else).
- If you cannot remove anything, it is time to either add **people** or dismiss clients.
- Technology may reduce the effort to do some monitoring, but that is often a **tricky gain** — did you reduce your burden after you introduced a time-series database in your infrastructure, or just won some more work to do ? Just asking for a friend . . .

Hygiene

I have some empirical rules to keep the environment healthy (and my sanity too).

- If you **add** some monitoring, then you to **remove** some monitoring (from somewhere else).
- If you cannot remove anything, it is time to either add **people** or dismiss clients.
- Technology may reduce the effort to do some monitoring, but that is often a **tricky gain** — did you reduce your burden after you introduced a time-series database in your infrastructure, or just won some more work to do ? Just asking for a friend . . .

Index

What can be wrong with sharing ?

- Surely sharing is a positive concept, with no drawback whatsoever.
- Yes, in theory.
- But in practice, you have to deal with:
  1. real people,
  2. regulations,
  3. agendas that are not aligned with yours (as is: somebody may be trying to steal your business).

What can be wrong with sharing ?

- Surely sharing is a positive concept, with no drawback whatsoever.
- Yes, in theory.
- But in practice, you have to deal with:
  1. real people,
  2. regulations,
  3. agendas that are not aligned with yours (as is: somebody may be trying to steal your business).

What can be wrong with sharing ?

- Surely sharing is a positive concept, with no drawback whatsoever.
- Yes, in theory.
- But in practice, you have to deal with:
  1. real people,
  2. regulations,
  3. agendas that are not aligned with yours (as is: somebody may be trying to steal your business).

What can be wrong with sharing ?

- Surely sharing is a positive concept, with no drawback whatsoever.
- Yes, in theory.
- But in practice, you have to deal with:
  1. real people,
  2. regulations,
  3. agendas that are not aligned with yours (as is: somebody may be trying to steal your business).

Trade secrets

It is not always a matter of protecting the recipe of your secret sauce, matters are more mundane.

- You could be required by *non-disclosure agreements* to keep mum about something.
- Do not underestimate the power of *security by obscurity*, in particular in very complex and messy environments.
- Being transparent (as in: showing all of your metrics to everybody) can work at your disadvantage, if you have dishonest partners.

Trade secrets

It is not always a matter of protecting the recipe of your secret sauce, matters are more mundane.

- You could be required by *non-disclosure agreements* to keep mum about something.
- Do not underestimate the power of *security by obscurity*, in particular in very complex and messy environments.
- Being transparent (as in: showing all of your metrics to everybody) can work at your disadvantage, if you have dishonest partners.

Trade secrets

It is not always a matter of protecting the recipe of your secret sauce, matters are more mundane.

- You could be required by *non-disclosure agreements* to keep mum about something.
- Do not underestimate the power of *security by obscurity*, in particular in very complex and messy environments.
- Being transparent (as in: showing all of your metrics to everybody) can work at your disadvantage, if you have dishonest partners.

Trade secrets

It is not always a matter of protecting the recipe of your secret sauce, matters are more mundane.

- You could be required by *non-disclosure agreements* to keep mum about something.
- Do not underestimate the power of *security by obscurity*, in particular in very complex and messy environments.
- Being transparent (as in: showing all of your metrics to everybody) can work at your disadvantage, if you have dishonest partners.

Shooting the messenger

- Last but not least, do not forget that automation and monitoring (this last, in particular) are a powerful way of **showing and visualizing** what happens.
- It could be that you will show things that would have better **be kept secret**.
- Sometime the one that gets shot for bad news is the messenger . . . try not to be on the **fire line**.

Shooting the messenger

- Last but not least, do not forget that automation and monitoring (this last, in particular) are a powerful way of **showing and visualizing** what happens.
- It could be that you will show things that would have better **be kept secret**.
- Sometime the one that gets shot for bad news is the messenger . . . try not to be on the **fire line**.

Shooting the messenger

- Last but not least, do not forget that automation and monitoring (this last, in particular) are a powerful way of **showing and visualizing** what happens.
- It could be that you will show things that would have better **be kept secret**.
- Sometime the one that gets shot for bad news is the messenger . . . try not to be on the **fire line**.

Shooting the messenger

- Last but not least, do not forget that automation and monitoring (this last, in particular) are a powerful way of **showing and visualizing** what happens.
- It could be that you will show things that would have better **be kept secret**.
- Sometime the one that gets shot for bad news is the messenger . . . try not to be on the **fire line**.

Index

Wrapping up

You are a programmer: keep calm Adopting DevOps techniques amounts to **create software** and software has its rules.

Democracy **No second hand citizens**: all the code has to be versioned, released, built, tested, documented, refactored, etcetera.

Own the process Infrastructure code, as anything, has **better and worse** tools, project, libraries, frameworks, etcetera.

Relax Corner cases do exists: learn to deal with them and when to **bend the rules**.

Wrapping up

You are a programmer: keep calm Adopting DevOps techniques amounts to **create software** and software has its rules.

Democracy **No second hand citizens**: all the code has to be versioned, released, built, tested, documented, refactored, etcetera.

Own the process Infrastructure code, as anything, has **better and worse** tools, project, libraries, frameworks, etcetera.

Relax Corner cases do exists: learn to deal with them and when to **bend the rules**.

Wrapping up

You are a programmer: keep calm
: Adopting DevOps techniques amounts to **create software** and software has its rules.

Democracy
: **No second hand citizens**: all the code has to be versioned, released, built, tested, documented, refactored, etcetera.

Own the process
: Infrastructure code, as anything, has **better and worse** tools, project, libraries, frameworks, etcetera.

Relax
: Corner cases do exists: learn to deal with them and when to **bend the rules**.

Wrapping up

You are a programmer: keep calm
: Adopting DevOps techniques amounts to **create software** and software has its rules.

Democracy
: **No second hand citizens**: all the code has to be versioned, released, built, tested, documented, refactored, etcetera.

Own the process
: Infrastructure code, as anything, has **better and worse** tools, project, libraries, frameworks, etcetera.

Relax
: Corner cases do exists: learn to deal with them and when to **bend the rules**.

Wrapping up

You are a programmer: keep calm
: Adopting DevOps techniques amounts to **create software** and software has its rules.

Democracy
: **No second hand citizens**: all the code has to be versioned, released, built, tested, documented, refactored, etcetera.

Own the process
: Infrastructure code, as anything, has **better and worse** tools, project, libraries, frameworks, etcetera.

Relax
: Corner cases do exists: learn to deal with them and when to **bend the rules**.

And remember . . .

SPAGHETTI ALLA BOLOGNESE . . .

DO. NOT. EXIST !

Thanks

Email m@biodec.com
Useless social @gaunilone
Incontro DevOps Italia See you in Bologna, March 5-6, 2020.
https://2020.incontrodevops.it/