Reproduce Anything, Anywhere

A GENERIC SIMULATION SUITE FOR TANGO CONTROL SYSTEMS

Agile Development
Continuous Delivery & Integration

The Goal is to deliver frequent **well-tested** small upgrades to the Control System, in order to continously improve the system while building confidence on operators.

Agile Development
Continuous Delivery & Integration

The Goal is to deliver frequent well-tested **small upgrades** to the Control System, in order to continously improve the system while building confidence on operators.

4 people / 108 CR applications:

Suse 11.1 => Debian9
Rpm => debian packaging
Tango 7 => Tango 9
Taurus 3 => Taurus 4
Polling => Events
HDB/TDB => HDB++
No IOC hardware change (yet)
Minimum change in Apps
NO SHUTDOWN TIME

Agile Development
Continuous Delivery & Integration

- Unit Testing
- Functional Testing
- Integration Testing
- Regression Testing

Unit testing

(cc) xkcd.com

Agile Development
Continuous Delivery & Integration

- Unit Testing (only for new stuff)
- Functional Testing (w/out HW?)
- Integration Testing (infrastructure?)
- Regression Testing (obsolescence!)

Then ... *does it really pay off?*

Agile Development
Continuous Delivery & Integration

But ... *does it really pay off?*

... Yes, but the overhead of testing must tend to 0 and/or provide added value

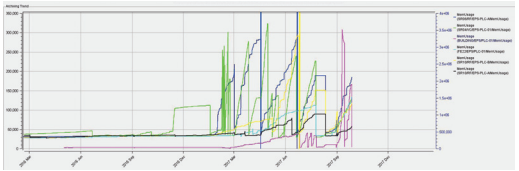
Testing Use Cases

While developing:

- Run real applications on test environment (w/out HW?)
- Regression tests, check for performance degradation

Continuous Integration (Jenkins, Travis, ...):

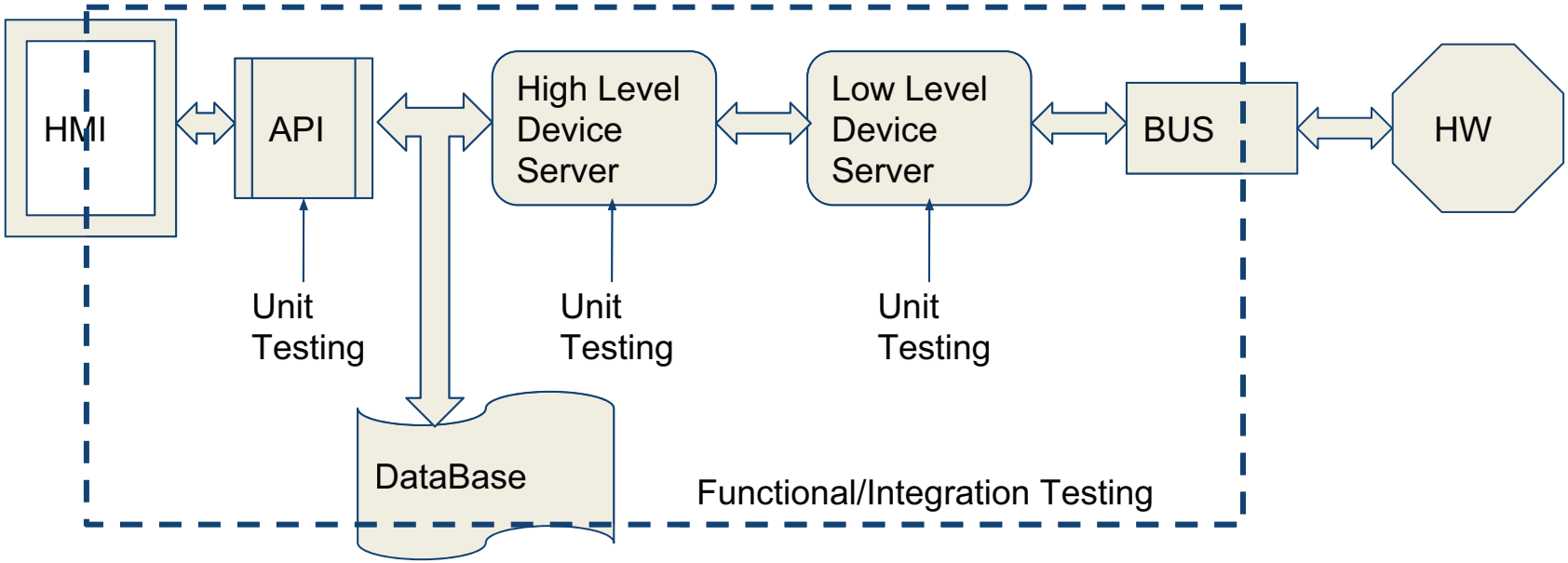
- Verify code correctness after each commit
- Perform Unit Testing
- Functional Testing against simulators
- Integration/Regression tests against dependencies



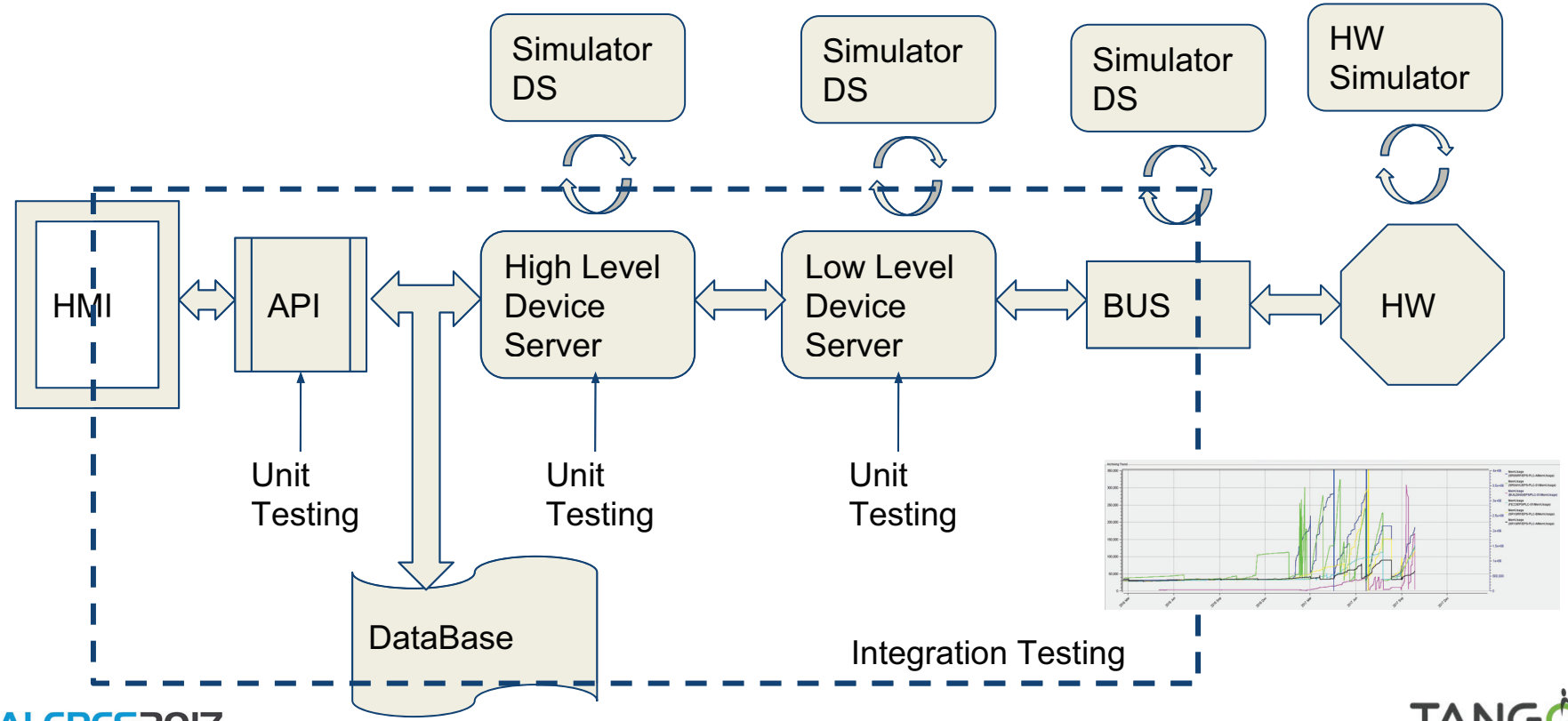
In production:

- Canary Testing, Keep an eye on performance!
- Bug found: Provide a replication environment (on Jira, Github?)
- Chaos Engineering: run experiments to characterize the system

What to Simulate?



The lower, the better



SimulatorDS

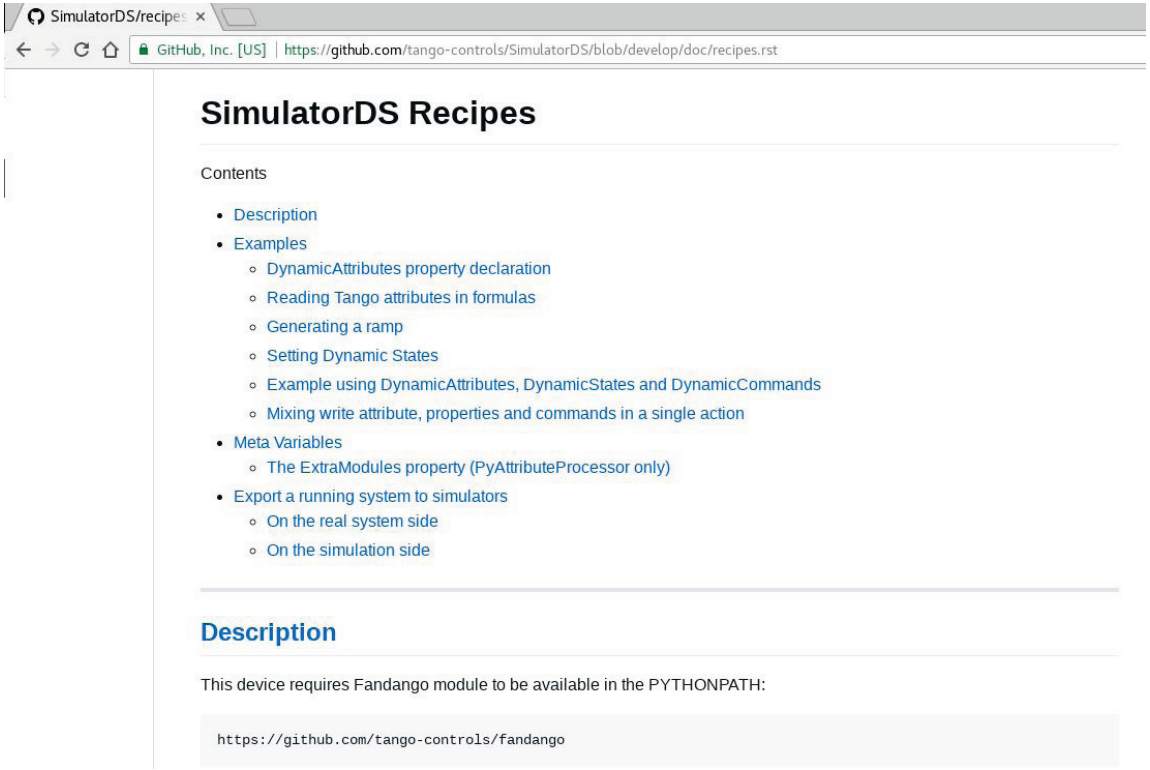
- An evolution of PySignalSimulator, developed at ALBA in 2007
- Simulation Code is stored in the Tango Database! (as Tango Device Properties)
- It consists of single-line python codes for each Tango attribute, state and command.

```
Input1=0.5+square(t) #(t = seconds since the device started)
Input2=2+1.5*sin(3*t)-0.5*random()
Buffer=DevVarLongArray([1.*sin(t*i) for i in range(1,10)])
```

- Used by ALBA, MAX IV, Solaris (and anyone using Vacca/Panic demo examples)
- Used for "Validation of a MySQL datababase for Tango Archiving", ICALEPCS'09

```
Square=0.5+square(t) #(t = seconds since the device started)
NoisySinus=2+1.5*sin(3*t)-0.5*random()
SomeNumbers=DevVarLongArray([1000.*i for i in range(1,10)])\

AVERAGE = DevDouble((XATTR('a/tango/device/attribute')+XATTR('other/tango/device/attribute'))/2.)

TEMPERATURE = 25+5*sin((1./60)*t)
PRESSURE = 2.5e-5+1e-6*sin((1./60)*t)
TEMPERATURES = DevVarDoubleArray([25+v*sin((1./k)*t) for v,k in [(5,60),(10,30),(15,45),(5,5)]])

MAX_TEMPERATURE = max(TEMPERATURES)
TEMP_LIMIT = READ and VAR('TEMP_LIMIT') or WRITE and VAR('TEMP_LIMIT',VALUE)

HOT = DevBoolean(MAX_TEMPERATURE>VAR('TEMP_LIMIT'))
COLD = DevBoolean(MAX_TEMPERATURE<15)
TEMPERATE = DevBoolean(not HOT and not COLD)
EXTREME = DevBoolean(HOT or COLD)
```

SimulatorDS



github.com/tango-controls/simulatords

SimulatorDS

The new SimulatorDS device was developed to:

- Move many dynamic features to fandango library, so more devices can be benefited of it (PyAttributeProcessor, WorkerDS, PyStateComposer, ...)
- Allow the final user to load custom libraries (e.g. database access).
- Allow templating, loading pre-defined constants, attributes, states and commands from python files.
- Include an scripting layer for simulation generation: *gen_simulation*

SimulatorDS

Device properties [test/sr/di]

Property name	Value
CheckDependencies	True
DynamicAttributes	CRef=float(HDB.get_attribute_values('sr/di/dcct/current',str2time('2017-03-01'),'+300')[N]) PRef=float(HDB.get_attribute_values('sr/vc/all/pressure',str2time('2017-03-01'),'+300')[N]) Noise=sin(PI*t) Pressure=PRef+1e-7*Noise Current=CRef+2*Noise
DynamicStates	ALARM=Pressure>1e-8 MOVING=0<Current<20 ON=Current>=20 OFF=1
ExtraModules	PyTangoArchiving.Reader as HDB lifetime_lib
KeepAttributes	True
KeepTime	500
LoadFromFile	/control/sr_di_calc.py
LogLevel	WARNING
UseEvents	false

SimulatorDS

```
CRef=float(HDB.get_attribute_values('sr/di/dcct/current',str2time('2017-03-01'),'+300')[N])
PRef=float(HDB.get_attribute_values('sr/vc/all/pressure',str2time('2017-03-01'),'+300')[N])
Noise=sin(PI*t)
Pressure=PRef+1e-7*Noise
Current=CRef+2*Noise
```

DynamicStates	ALARM=Pressure>1e-8 MOVING=0<Current<20 ON=Current>=20 OFF=1

Device properties [test/sr/di]

Property name	Value
CheckDependencies	True
ExtraModules	PyTangoArchiving.Reader as HDB lifetime_lib
IgnoreList	
KeepAttributes	True
KeepTime	500
LoadFromFile	/control/sr_di_calc.py
LogLevel	WARNING
SortLists	True
UseEvents	false
UseTaurus	False

```
##############################################################################
#                              SIMULATION                                    #
##############################################################################

#                           Constant Parameters                              #
##############################################################################
Uo = 1
Uc = 1.1
Rs = 3.3e6
Qo = 29500
PI = 3.1415927
Working_Frequency = 499650000
B06A = 2.701
B06B = 2.687
B10A = 2.4582
B10B = 2.5467
B14A = 2.678
B14B = 2.5639

#                        Select ID open or ID closed                         #
##############################################################################
# Flag to control if the IDs are open: True when open; False when closed
idsOpen = bool(VAR('idsOpen',VALUE) if WRITE else VAR('idsOpen') or False)
U = Uo if idsOpen else Uc

#                              Select Ibeam                                  #
##############################################################################
machinebeam = XATTR('sr/di/dcct/averagecurrent')
usrDefIbeam = float(VAR('UserCurrent',VALUE) if WRITE else VAR('UserCurrent',default=130))
setUsrDefIbeam = bool(VAR('usrIbeam',VALUE) if WRITE else VAR('usrIbeam', default=False))
Ibeam = usrDefIbeam if setUsrDefIbeam else machinebeam
I = Ibeam/1000
```

gen_simulation

This script, part of SimulatorDS package, provides guidance to the developer to create a simulation from an existing system:

- Gets the list of attributes being accessed by a PyTango process.
- Exports devices/attributes values, buffers and configuration to CSV, pickle or JSON files.
- Generate a default SimulatorDS template for each exported Device class.
- Reload the SimulatorDS templates on a new Tango Host (MySQL or SQLite).
- Apply the events/polling/host/runlevel configuration for all the exported devices.
- Start/stop the device servers with the desired topology

Using .csv? In 2017?

We use CSV spreadsheets as a support format for import/export between systems because:

- It is human readable and more compact than JSON and XML
- Unlike other DSL, we are specifying different device behaviours in a single file
- It is very hard to extend, but on the other hand it will keep it simple.
- At the end, the code is really inserted and kept in database.

A	B	C	D	E	
host	server	class	device	property	value
Tango-chaos-4	DynamicDS/elinac-beam	SimulatorDS	elin/beam/run	DynamicAttributes	Cleaning = DevDouble(VAR('Cleaning',default=True * (1+2*sin((int(PROPERTY("OFFSET
					DeflectionDC = DevDouble(VAR('DeflectionDC',default=0.0 * (1+2*sin((int(PROPERTY("
					Delay = DevDouble(VAR('Delay',default=6.93217931773e-310 * (1+2*sin((int(PROPERTY
					GridV = DevDouble(VAR('GridV',default=-72.0 * (1+2*sin((int(PROPERTY("OFFSET"))+t
					PulseL = DevDouble(VAR('PulseL',default=2.1 * (1+2*sin((int(PROPERTY("OFFSET"))+t
					PulseType = DevDouble(VAR('PulseType',default=SHORT * (1+2*sin((int(PROPERTY("C
					PulseV = DevDouble(VAR('PulseV',default=15.6 * (1+2*sin((int(PROPERTY("OFFSET")
				DynamicCommands	Off = DevString('elin/beam/run/Off')
					On = DevString('elin/beam/run/On')
					Reset = DevString('elin/beam/run/Reset')
				DynamicStates	MOVING=t%randint(15,30)>randint(1,15)
					ALARM=t%randint(1,30)<randint(1,6)
					ON=1
				LoadFromFile	/home/srubio/CO/vacca/examples/elinac/LinacGun_attributes.txt
				OFFSET	
Tango-chaos-5	DynamicDS/elinac-cool	SimulatorDS	elin/cool/1	DynamicAttributes	BuncherTemp = DevDouble(VAR('BuncherTemp',default=27.1 * (1+2*sin((int(PROPERTY
					PBuncherTemp = DevDouble(VAR('PBuncherTemp',default=26.1 * (1+2*sin((int(PROPER
					SectionTemp = DevDouble(VAR('SectionTemp',default=28.1 * (1+2*sin((int(PROPERTY('
					WaterTemp = DevDouble(VAR('WaterTemp',default=25.1 * (1+2*sin((int(PROPERTY("OF
				DynamicCommands	Reset = DevString('elin/cool/1/Reset')
				DynamicStates	MOVING=t%randint(15,30)>randint(1,15)
					ALARM=t%randint(1,30)<randint(1,6)
					ON=1
				LoadFromFile	/home/srubio/CO/vacca/examples/elinac/LinacCooling_attributes.txt
				OFFSET	
Tango-chaos-5	DynamicDS/elinac-cool	SimulatorDS	elin/cool/bun-temps	DynamicAttributes	Current = DevDouble(VAR('Current',default=223.0 * (1+2*sin((int(PROPERTY("OFFSET"
					Frequency = DevDouble(VAR('Frequency',default=3140.0 * (1+2*sin((int(PROPERTY("O
					Interlocks = DevVarDoubleArray(VAR('Interlocks',default=[0 * (1+2*sin((int(PROPERTY('
					Position = DevDouble(VAR('Position',default=54.5 * (1+2*sin((int(PROPERTY("OFFSET

Current uses of SimulatorDS

- Vacca (SCADA application for Tango) demo / test suite.
- PANIC (SCADA application for Tango) demo / test suite.
- Scale Tests on Tango Archiving on MySQL (HDB/TDB and HDB++).
- Event/polling tuning tests and benchmarks for Taurus 4/Tango 9 applications.
- Simulation of RF, Vacuum, EPS and BL29 subsystems at ALBA, for debugging and operator training.
- Device Server Continous Integration (PyPLC, Sardana, PyAlarm).
- Bug reproducibility
- "*Chaos Engineering*" tests on AWS Cloud (ESRF).

PyAlarm/Clock	PyAlarm	test/panic/ck02	AlarmList	CK02:False and not CK02
			AlarmSeverities	CK02:DEBUG
			AlarmThreshold	1
		test/panic/ck05	AlarmList	CK05: False and not CK05
			AlarmSeverities	CK05:DEBUG
			AlarmThreshold	2
			PollingPeriod	0.125
		test/panic/ck1	AlarmList	CK1: not CK1
			AlarmSeverities	CK1:DEBUG
			AlarmThreshold	5
		test/panic/ck2	AlarmList	CK2: NOW()%2 < 1
			AlarmSeverities	CK2:DEBUG
			AlarmThreshold	1
		test/panic/ck5	AlarmList	CK5: NOW()%5<2.5
			AlarmSeverities	CK5:DEBUG
			AlarmThreshold	1
		test/panic/ck10	AlarmList	CK10:NOW()%10<5
			AlarmSeverities	CK10:DEBUG
			AlarmThreshold	1
PyAlarm/Group	PyAlarm	test/panic/group	AlarmList	GROUP_0: GROUP(test/panic/ck*/ck5|ck10)
				GROUP_1: GROUP(CK5,CK10)
				GROUP_OR: GROUP(CK5,CK10,x>=1)
				GROUP_AND: test/panic/ck2/ck2 and test/panic/ck5/ck5
				GROUP_ALL: GROUP(GROUP_0,GROUP_1,GROUP_2,GROUP_AND)
			AlarmSeverities	GROUP_1: ALARM
				GROUP_OR: ALARM
				GROUP_AND: ALARM
				GROUP_ALL:ALARM
			AutoReset	0.0001
			PollingPeriod	0.5
			AlarmThreshold	1

Chaos Engineering?

Experiments in production?

Canary Testing?

principlesofchaos.org

Build a Hypothesis around Steady State Behavior

Focus on the measurable output of a system, rather than internal attributes of the system. Measurements of that output over a short period of time constitute a proxy for the system's steady state. The overall system's throughput, error rates, latency percentiles, etc. could all be metrics of interest representing steady state behavior. By focusing on systemic behavior patterns during experiments, Chaos verifies that the system *does* work, rather than trying to validate *how* it works.

Vary Real-world Events

Chaos variables reflect real-world events. Prioritize events either by potential impact or estimated frequency. Consider events that correspond to hardware failures like servers dying, software failures like malformed responses, and non-failure events like a spike in traffic or a scaling event. Any event capable of disrupting steady state is a potential variable in a Chaos experiment.

Run Experiments in Production

Systems behave differently depending on environment and traffic patterns. Since the behavior of utilization can change at any time, sampling real traffic is the only way to reliably capture the request path. To guarantee both authenticity of the way in which the system is exercised and relevance to the current deployed system, Chaos strongly prefers to experiment directly on production traffic.

Automate Experiments to Run Continuously

Running experiments manually is labor-intensive and ultimately unsustainable. Automate experiments and run them continuously. Chaos Engineering builds automation into the system to drive both orchestration and analysis.

Minimize Blast Radius

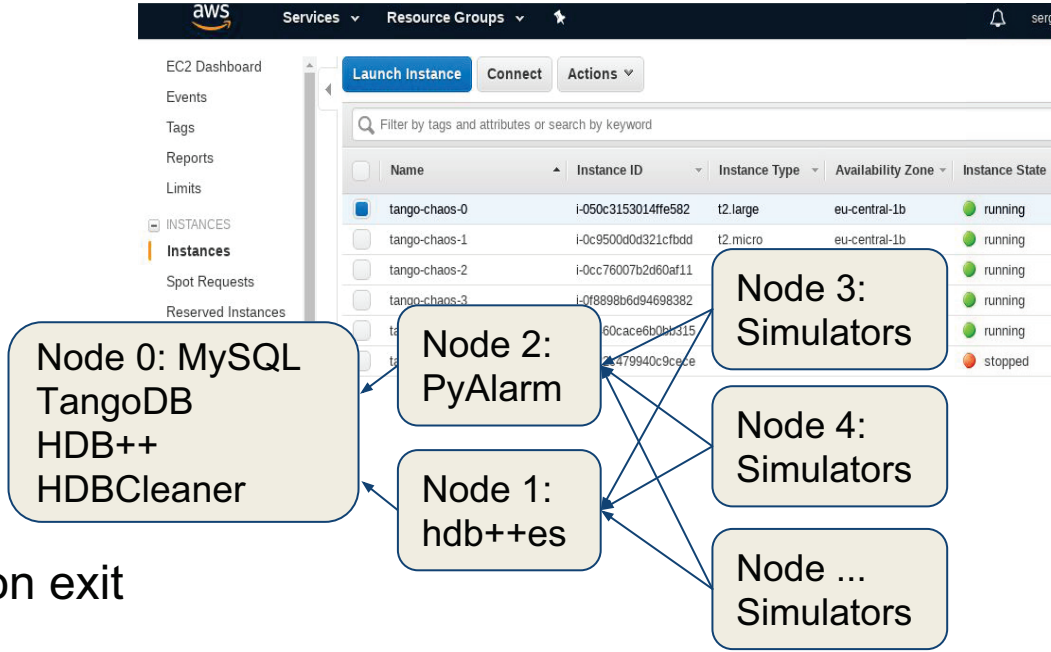
Experimenting in production has the potential to cause unnecessary customer pain. While there must be an allowance for some short-term negative impact, it is the responsibility and obligation of the Chaos Engineer to ensure the fallout from experiments are minimized and contained.

Chaos Tests on AWS

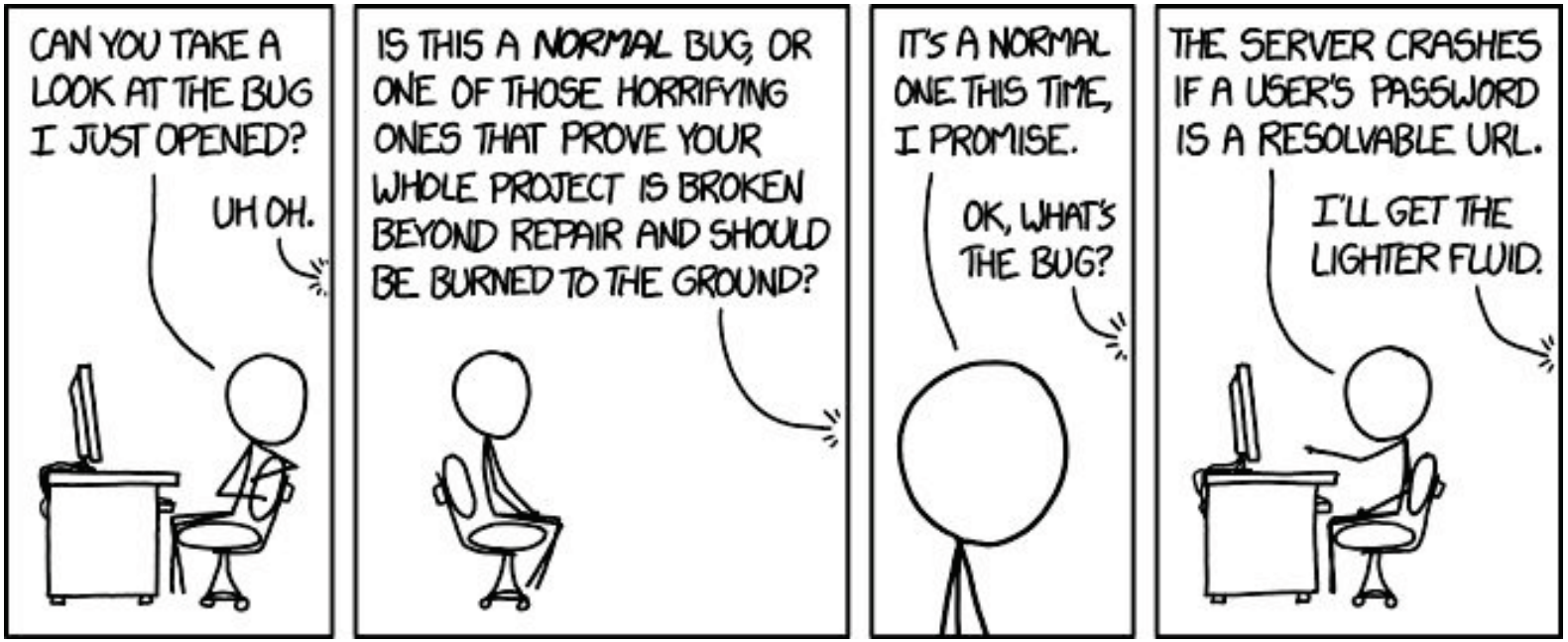
Andy Götz proposed (and helped) to create a Tango testing platform on AWS

We developed scripts to:

- export a control system to simple .csv files
- simplify aws-cli usage
- start/create/list instances
- obtain public and private DNS
- create and configure devices
- modify device setup easily
- export and delete hdb++ database on exit



Bug Reproducibility



(cc) xkcd.com

Bug Reproducibility

- A bug is found in X project, but it's only reproduceable in our local system.
- The X software is community-supported in github or somewhere, by developers that have no access to our local setup to reproduce it.
- *gen_simulation* allows to attach devices and attributes configurations in csv files (including its event/polling behaviour and some recorded data).
- device/servers distribution in hosts can also be included.
- SimulatorDS enables the developer to reproduce the reported bug in containers or cloud ... or not, in which case we can discard X as the cause.

Test Case

```
$ tango2csv test/tango/rw rw.csv

$ csv2tango rw.csv

$ tango_servers stop test/tango/rw
$ tango_servers start test/tango/rw

$ ipython
: from PyTango import DeviceProxy
: dp = DeviceProxy ('test/tango/rw')
: dp.write_attributes(
    [('a',2),('b',20)])
            DevFailed: DevFailed[
            DevError[
               desc = Set value for attribute B is above the maximum authorized (at least element 0)
             origin = WAttribute::check_written_value()
             reason = API_WAttrOutsideLimit
            severity = ERR]
: [v.value for v in dp.read_attributes(['a','b'])]
Out:: [2.0, 0.0]
```

	A	B	C	D	E
1	server	class	device	property	value
2	SimulatorDS/testrw	SimulatorDS	test/tango/rw	DynamicAttributes	A=VAR("A",WRITE=True,default=0)
3					B=VAR("B",WRITE=True,default=0)
4				B.min_value	0
5					

	A	B	C	D	E
1	server	class	device	property	value
2	SimulatorDS/testrw	SimulatorDS	test/tango/rw	DynamicAttributes	A=VAR("A",WRITE=True,default=0)
3					B=VAR("B",WRITE=True,default=0)
4				B.min_value	0
5				B.max_value	16
6					

Thanks for your attention