CS NANO

CS GROUP

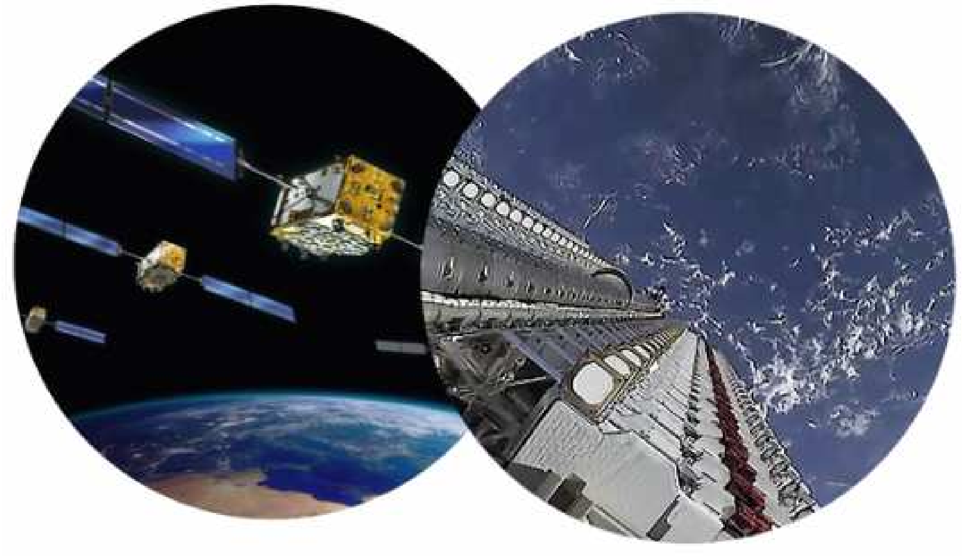
CCnode

Innovative Operational Software Architecture To Address Large Constellations Of Satellites

Ground Systems Architecture Workshop (GSAW) – 2021

INTELLIGENT & CYBER-PROTECTED MISSION-CRITICAL SYSTEMS

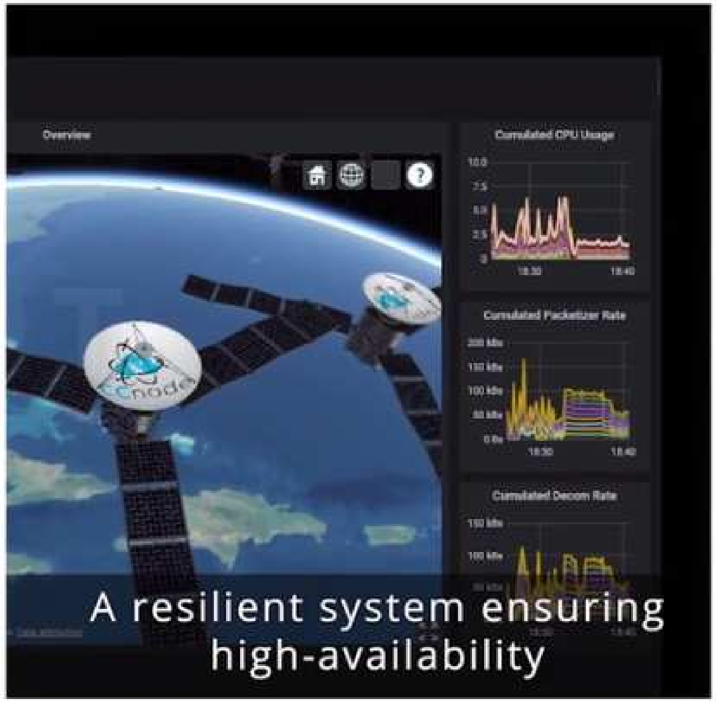
New Needs Of Constellations



- Critical needs for constellations control centers
  - ❑ Performance
  - ❑ Scalability
  - ❑ Automation
  - ❑ High-level of Availability
- CS GROUP solution : full microservices architecture

Microservices Architecture

- Provides solutions
  … but imposes new constraints
- New architecture paradigms
  - ❑ Past : large servers to process telemetry and telecommands
  - ❑ Present / future : microservices
- Available data on microservices not really usable (theoretical, oversimplified)
- Our deal for CS Nano : build a real microservices architecture which handles constellations constraints by design



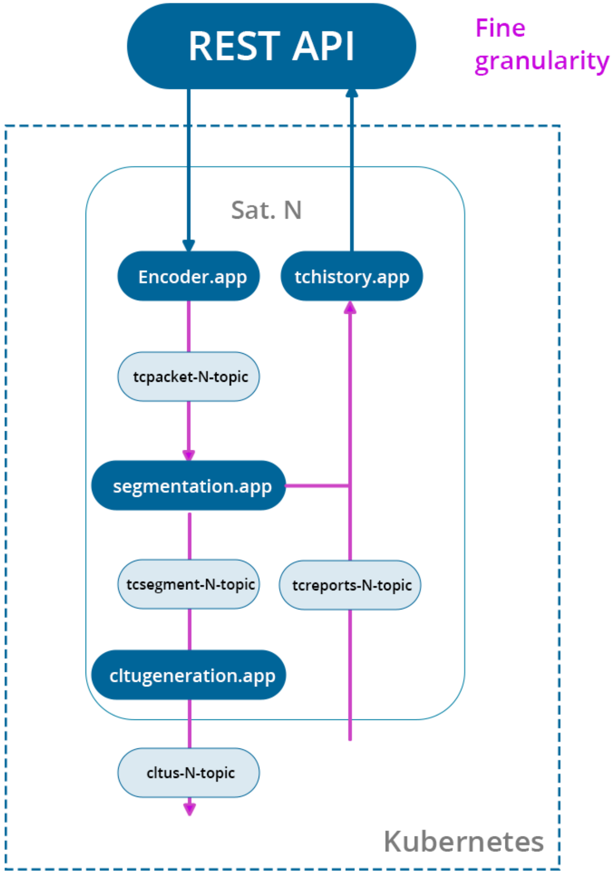
A resilient system ensuring high-availability

SCALABILITY

- Existing systems were not built to specifically address constellations
- Large constellations require architecture and software scalability
  - ❑ Increasing / variable number of satellites
  - ❑ Variable computation power
- Adaptations on existing control centers are inadequate
- Scalability must be adressed for computing power and operability
- Our work: optimize microservices granularity and resources management

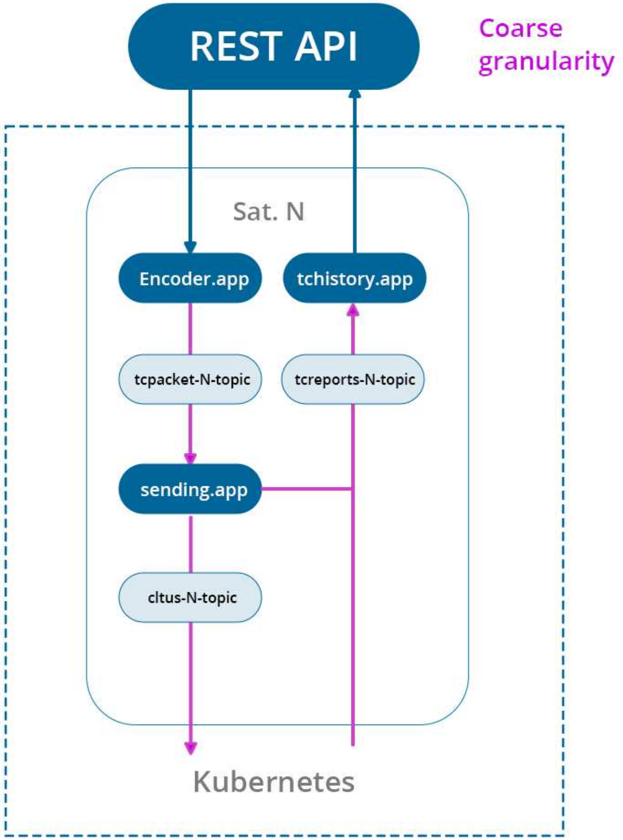
MICROSERVICES GRANULARITY

- Choice of microservices granularity is an important issue
- Too fine granularity may lead to
  - ❑ Too many data exchanges
  - ❑ Out of control complexity
  - ❑ Unintended overload



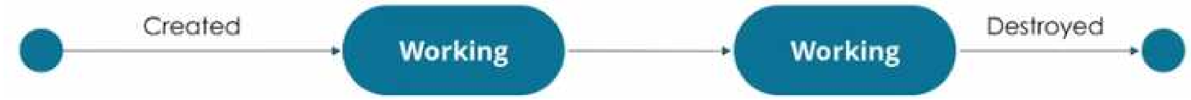
MICROSERVICES GRANULARITY

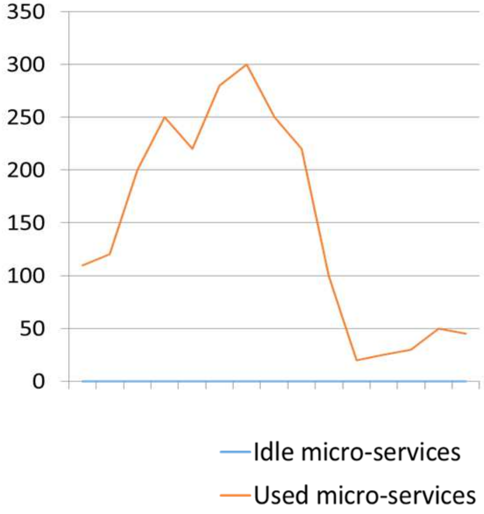
- ➢ Choice of microservices granularity is an important issue
- ➢ Too fine granularity may lead to
  - ❑ Too many data exchanges
  - ❑ Out of control complexity
  - ❑ Unintended overload
- ➢ Coarse granularity helps reducing the number of processes
- ➢ Our work: find the right balance depending on functions characteristics



RESOURCES MANAGEMENT

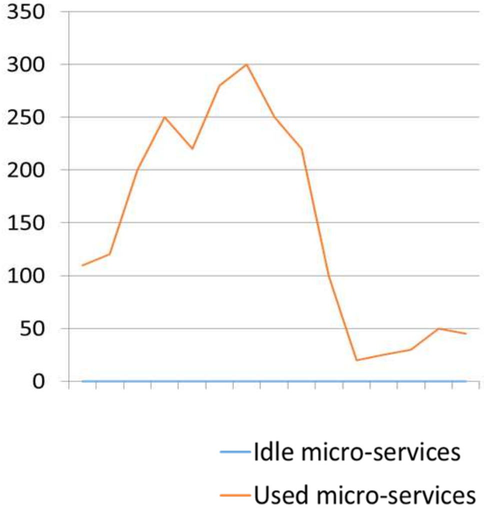
- Meet scalability requirements requires a resource management strategy
- 1st strategy: create resource « on-demand »
  - ❑ Adjust resources allocation dynamically
  - ❑ Microservices have resources when they are used

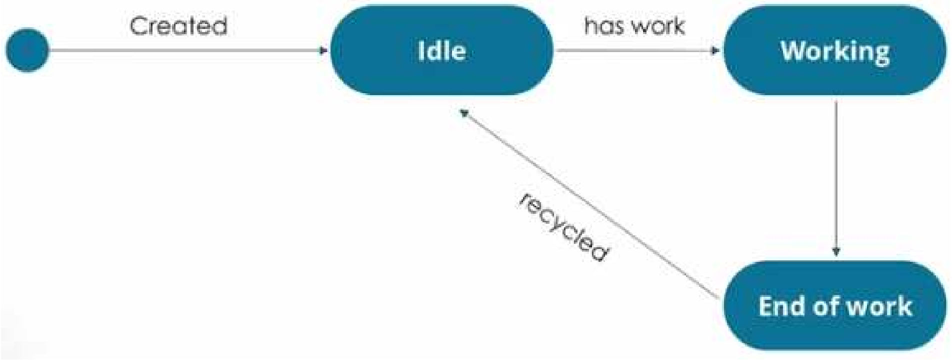




RESOURCES MANAGEMENT

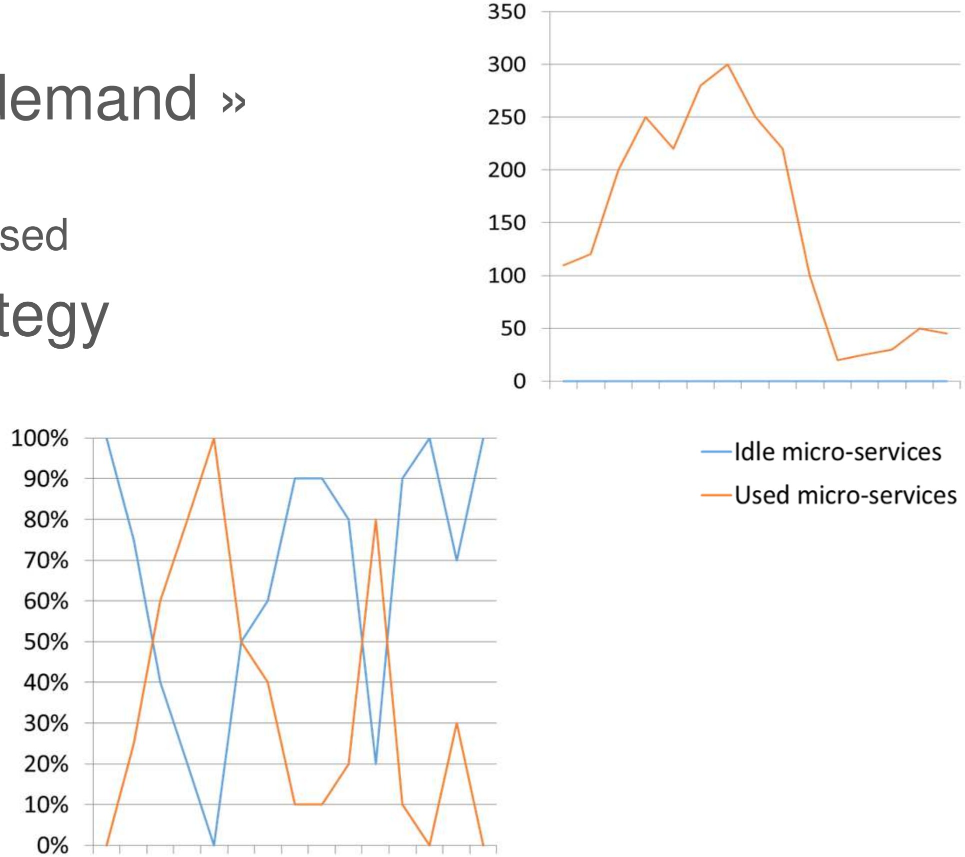
- Meet scalability requirements requires a resource management strategy
- 1st strategy: create resource « on-demand »
  - ❑ Adjust resources allocation dynamically
  - ❑ Microservices have resources when they are used
- 2nd strategy: use a « booking » strategy





Resources Management

- Meet scalability requirements requires a resource management strategy
- 1st strategy: create resource « on-demand »
  - ❑ Adjust resources allocation dynamically
  - ❑ Microservices have resources when they are used
- 2nd strategy: use a « booking » strategy
  - ❑ Create all microservices at system startup
  - ❑ Recycle microservices which are no longer used



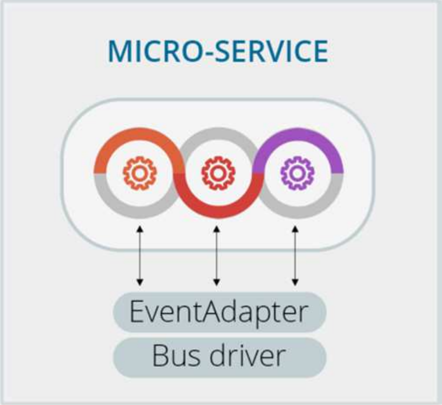
Resources strategy comparison

	On-demand	Booking
Overhead	Important	None
Reactivity	Microservices must start quickly	Better
Scalability	Dynamic	Static
Process number	Dynamic	Static
Resources allocation	Adjusted, optimized	Maximal
System stability	Lots of variations	Better
Microservice complexity	Simple	Complex (recycling, elections)

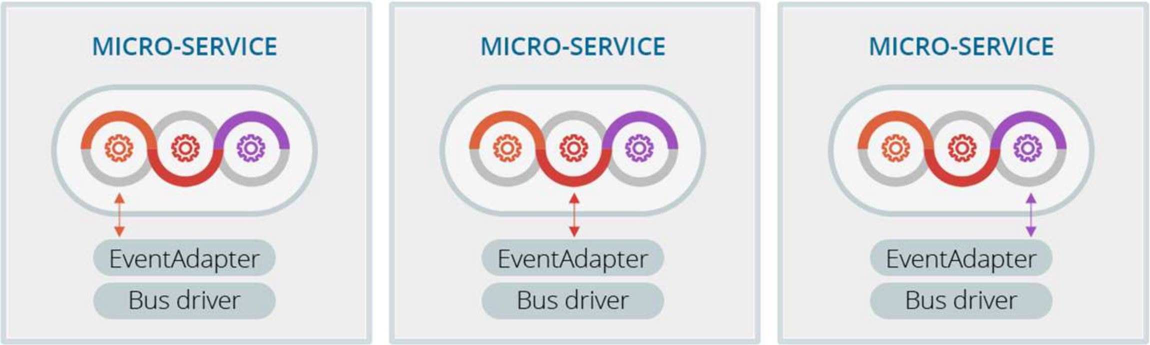
Legacy Code Integration

- Reuse existing tested and operational components: often done for economic or schedule reasons

- Macroservice black-box
  - ❑ Component not changed
  - ❑ Very pragmatic, but violates good practices



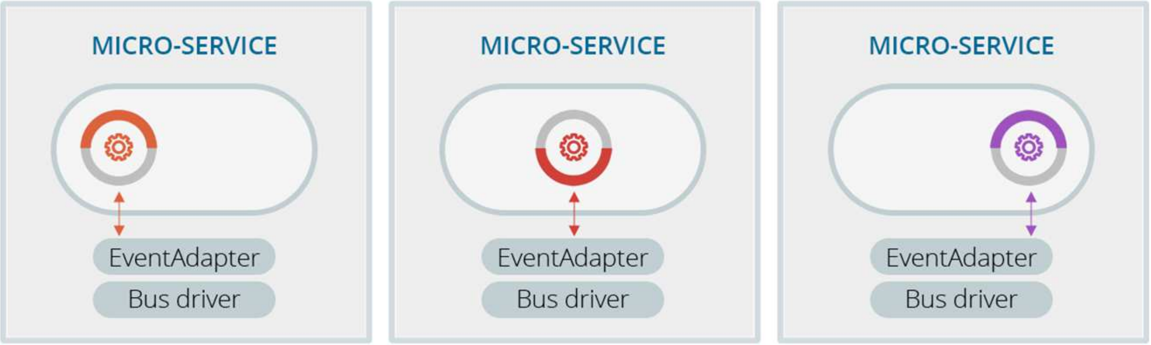
- Microservice black-box
  - ❑ A microservice per function
  - ❑ Real microservice but bound to original complexity



LEGACY CODE INTEGRATION

Microservice white box

- ❑ When possible to extract functions into independent modules
- ❑ Results and benefits
  - Limited amount of code
  - Limited dependencies
  - Clean API available to use modules
- ❑ Real microservices in depth



LEGACY CODE INTEGRATION

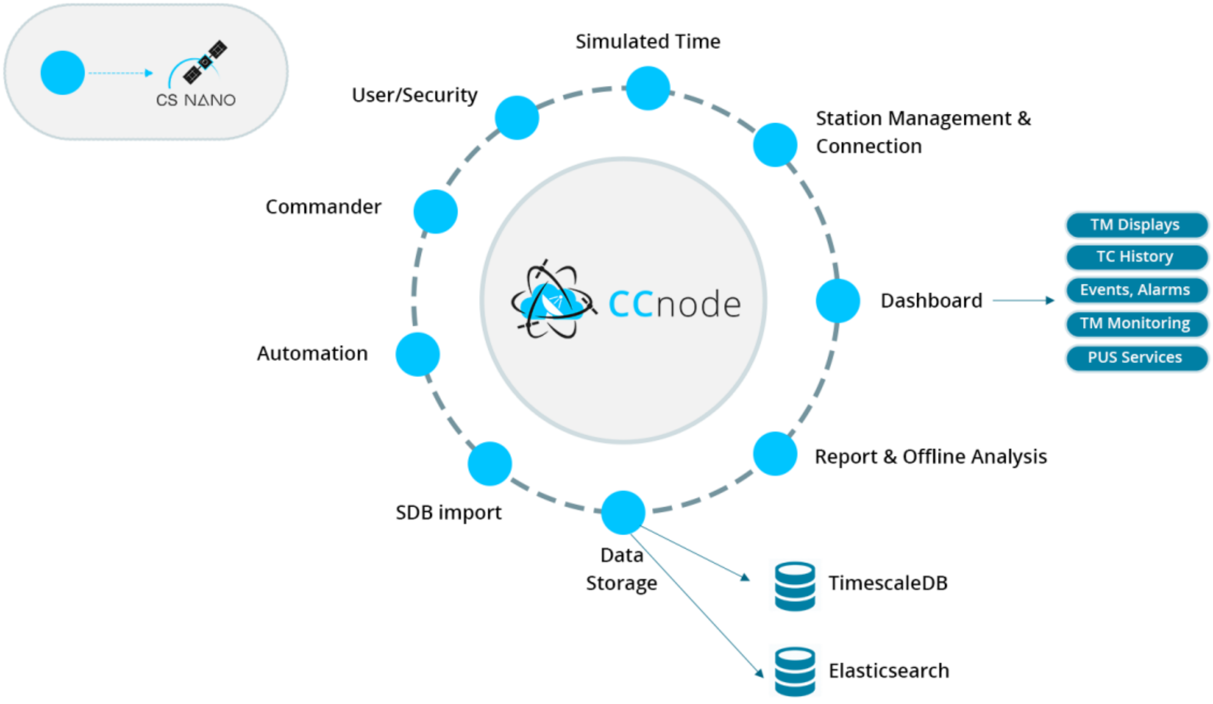
	Macroservice black box	Microservice black box	Microservice white box
Type of work	Pragmatic interfaces development	Little work	A lot of work
Containerization	Complex internal state (several processes in the same container), impossibility to manage life cycles of different logical components		Good because of single process container
Extraction from legacy systems	No	No	Yes
Legacy systems refactoring	No	No	Yes
Respect to microservices	No	Only on the surface	Yes

Should be avoided

Good compromise
Gradual migration

Best choice if time and money

PUTTING IT ALL TOGETHER - CCNODE

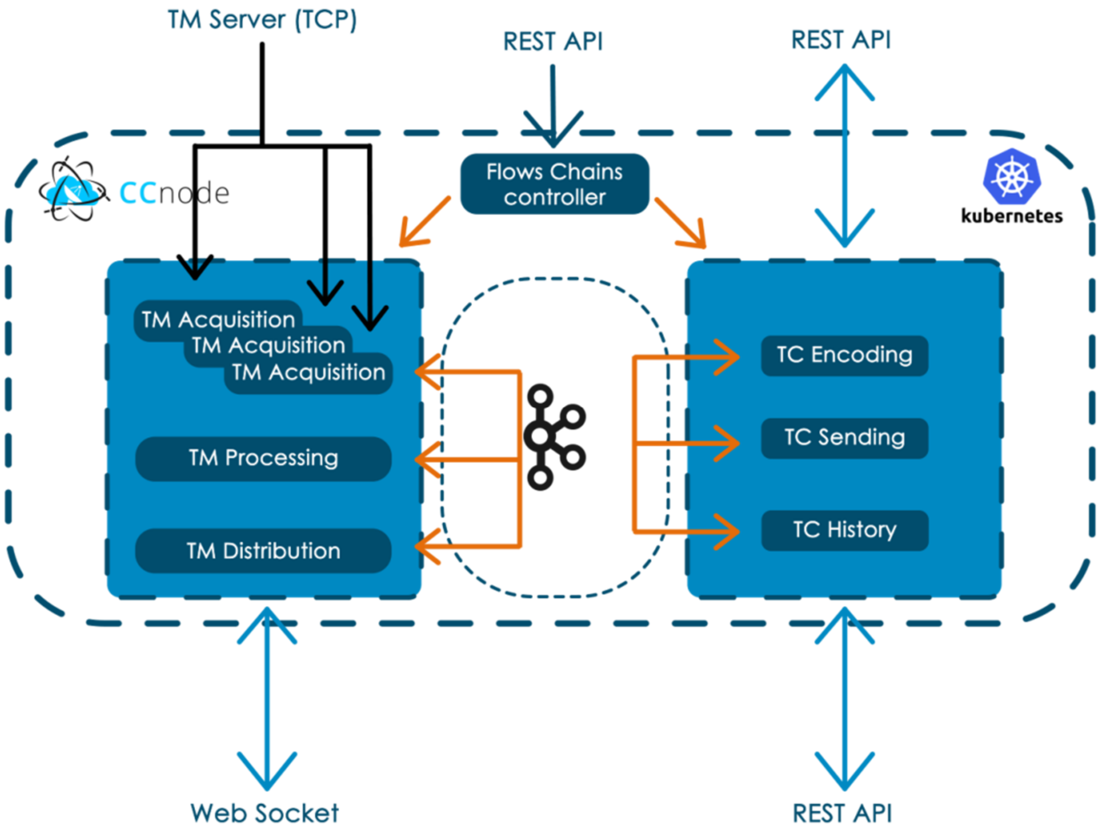


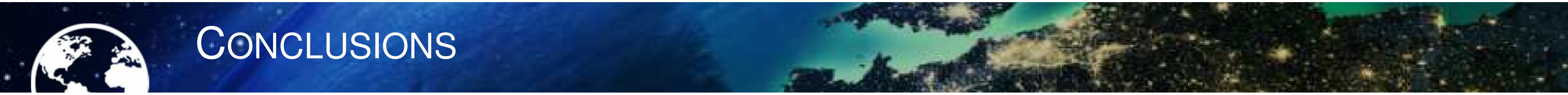
Microservices are

- The most simple as possible
- Developed without OS adherence
- Stateless / Stored in a highly-available storage
- Delivered as Docker images, orchestrated by Kubernetes
- Equipped with uniform logs collected in ElasticStack
- Organized as stacks into helm charts
- Accessed externally via APIs (REST or Websocket)
- Loosely-coupled to enable high-availability

CCnode Key Aspects

- Updating services without interruption of services
- Autonomous
- Highly-resilient by nature
- Scalable and flexible
  - Thanks to Kubernetes / Kafka
  - Process as many satellites as necessary if resources are available
- Security aware
- Code reuse
- Economically Cloud-ready





CONCLUSIONS

- CS Nano / CCnode: ready to be integrated in a constellation Monitoring & Control System
- CS Nano product line: based on CS GROUP strong experience in satellites operations and investments in R&D activities.
- CS Nano / CCnode: a pragmatic, robust, simple, performant and modern solution, which is cloud-ready, scalable and flexible.
- In 2021:
  - Prototyping activities in ESA B2-phase programme
  - Tested on an operational nanosatellite mission led by CNES
- CCnode connectivity allows it to be used in many New Space missions (SSA projects for example).

CS NANO

CS GROUP

CCnode

THANK YOU FOR YOUR ATTENTION!

QUESTIONS WELCOME AT CSNANO@CSGROUP.EU

Authors: Pierre Bornuat, Pierre-Alban Cros, Christophe Pipo, Sun Volland (CS GROUP – France)