

# .NET Conf CL


## v2018



#EXPLOTA

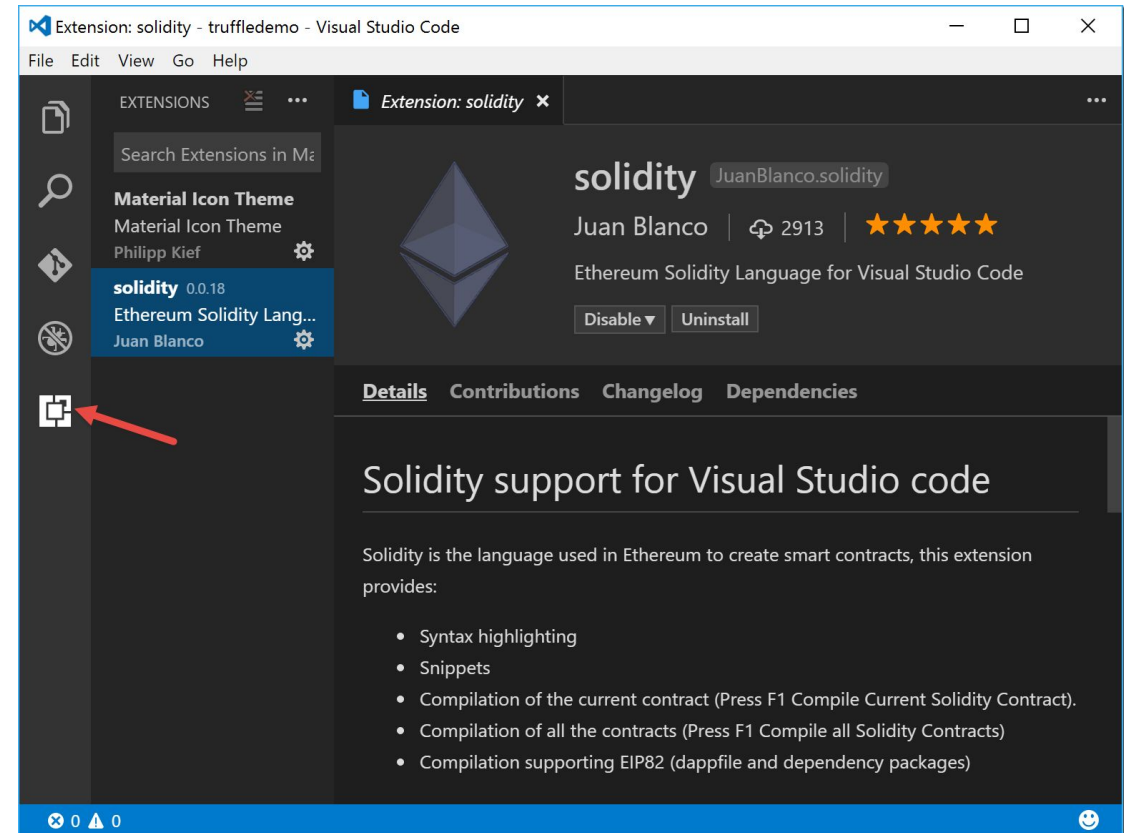
Trabajando con Azure Blockchain y .NET Core

## Sebastián Pérez

 @sebaleoperez

# Requerimientos

- Java JDK 1.8
- Visual Studio Code
- Solidity Plugin (Juan Blanco)
- .NET Core
- Nethereum (NuGet)



# Blockchain

introducción

# Blockchain

- Transparencia
- Inmutabilidad
- Bajo Costo
- Transacciones rápidas
- Seguridad





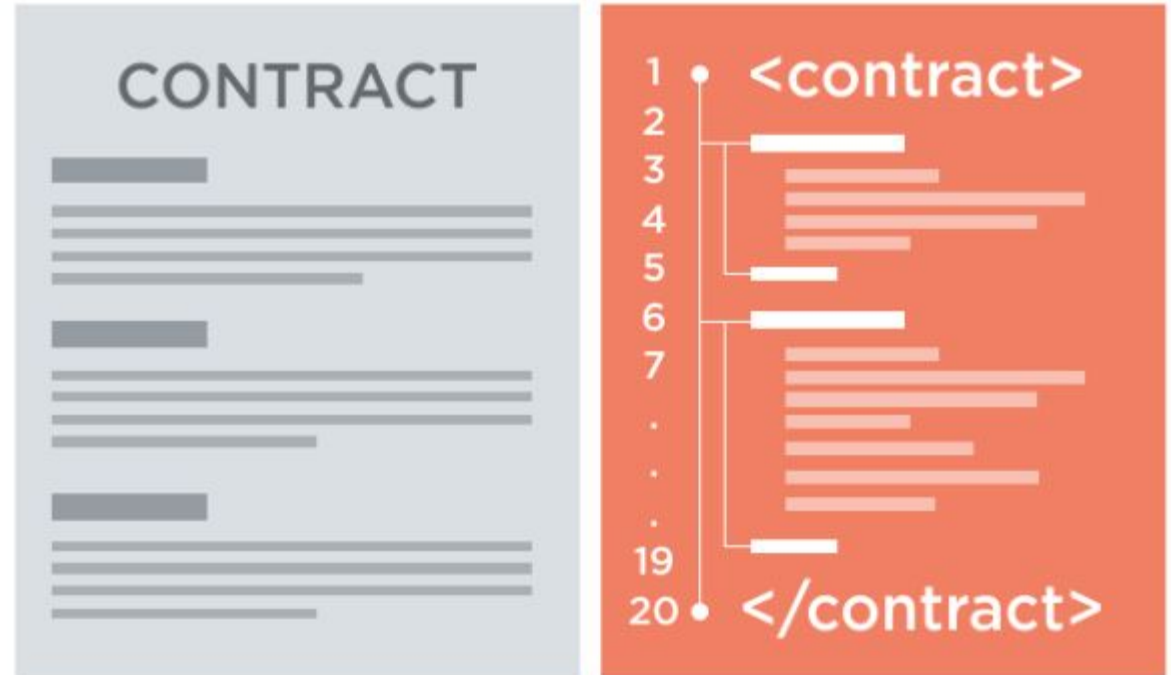
# Bitcoin

- Transparencia
- Inmutabilidad
- Bajo Costo
- Transacciones rápidas
- Seguridad



# Contratos inteligentes

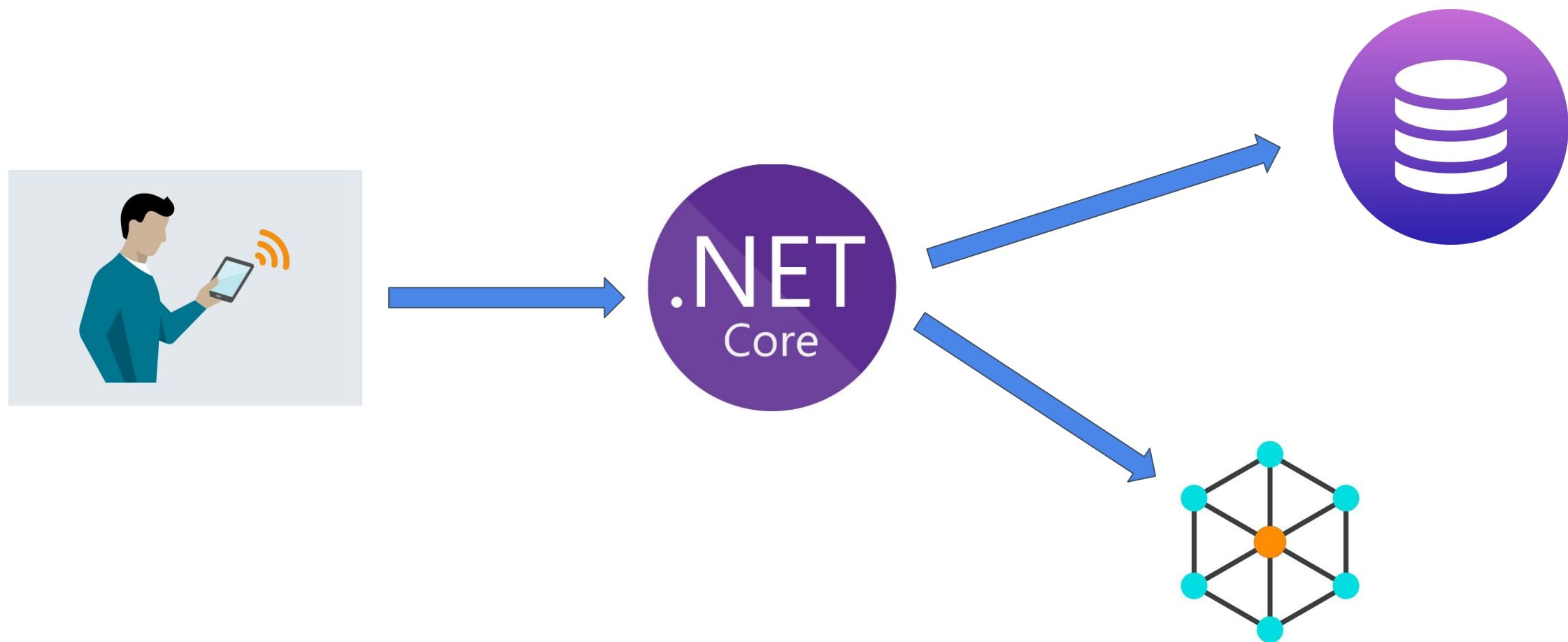
- Transparencia
- Inmutabilidad
- Bajo Costo
- Transacciones rápidas
- Seguridad
- Auto – Verificable
- Auto – Ejecutable
- Conciliación Automática



# Preparando el entorno

Desarrollando Smart Contracts

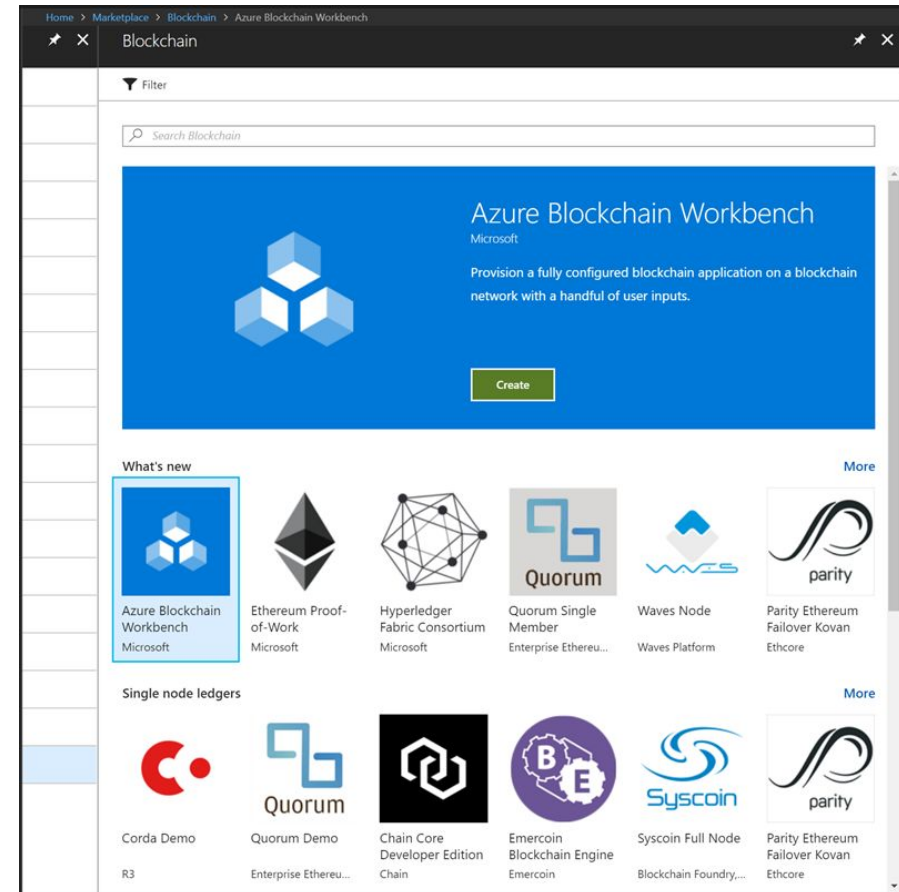
# Arquitectura de una Dapp



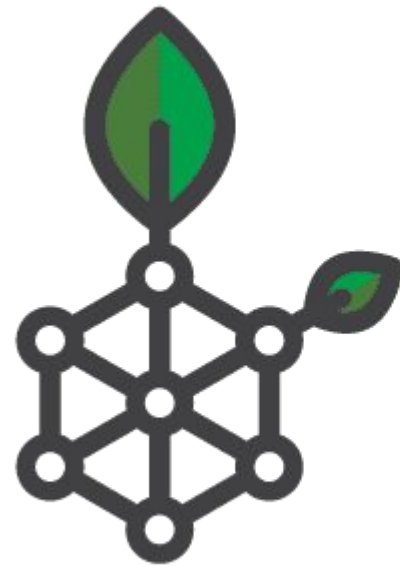


# Infraestructura

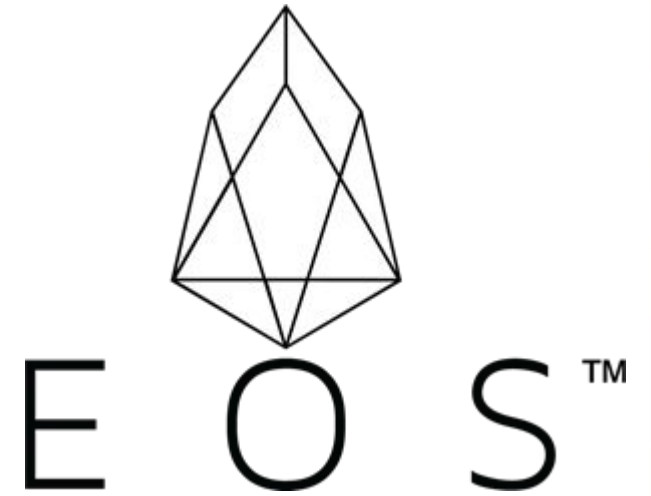
- Nodos Mainnet, Testnet, Regtest
- Azure Marketplace > Blockchain
- Azure Blockchain Workbench



# Infraestructura



RSK



# RSK

- EVM 100% compatible con Ethereum
- Soporta un volumen alto de transacciones a bajo costo
- Alta seguridad
- Contratos costeados en Bitcoin
- <http://rsk.co>



# Crear un nodo de RSK en Azure

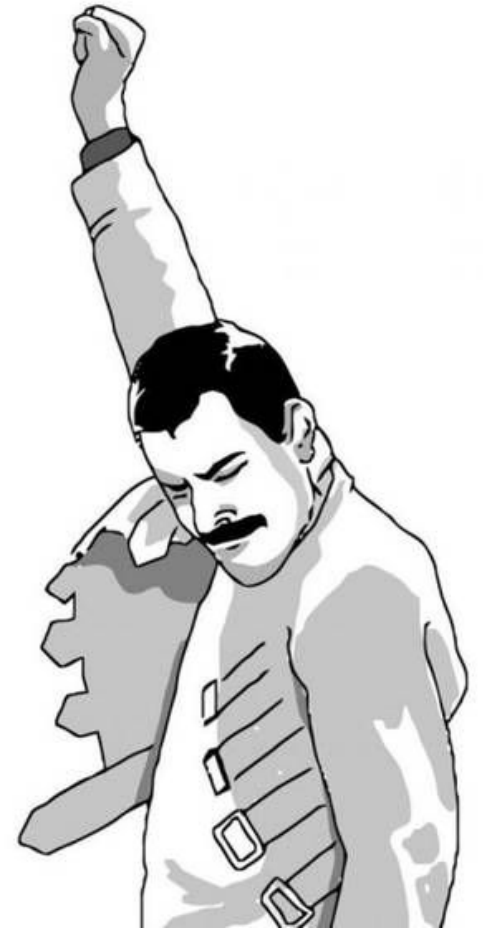


# Crear un nodo local - regtest

- Entrar a <https://github.com/rksmart/rskj/releases>
- Bajar la última versión (0.5.3 - ORCHID)
- Abrir una terminal en la carpeta donde bajamos el jar
- Correr el siguiente comando:
  - `java -cp ./rskj-core-0.5.3-ORCHID-all.jar co.rsk.Start --regtest`

# Corroborar que todo esté bien

- Bajar el proyecto RSK Manager de mi github
  - <http://github.com/garudaslap>
- Abrirlo con Visual Studio
- Ejecutar el test “**GetAccountsTest**”







# Let's Work !

Manos a la obra !

# Chequear si hay fondos disponibles

- Corroboraremos que existe la cuenta “**Cow**” con el Test “**CheckIfCowAccountExists**”
- Corroboraremos si tiene saldo con “**CheckBalanceOfCowAccount**”



# Crearemos una nueva cuenta

- Llamaremos al método “**CreateNewAccount**” poniendo un breakpoint en la última línea del Test para ver la información de la cuenta
- Guardaremos el address y la clave privada en los settings
- Corroboraremos el saldo 0 de la nueva cuenta con “**CheckBalanceOfCreatedAccount**”



# Daremos saldo a la cuenta

- Enviaremos saldo a la cuenta con el método **“SendGasFromCow...”**
- Corroboraremos si tiene saldo ahora con **“CheckNewBalance...”**



# Crearemos un contrato

- Abriremos el contrato **HelloWorld** del repositorio con Visual Studio Code
- Presionamos “**F1**” y buscaremos “**compile current contract**”
- Aparecerá en la carpeta “**bin**” la compilación, de ahí sacaremos el bytecode y el abi para guardarlos en los settings
- Llamaremos al método “**DeployContract**” del Test y con un breakpoint en la salida chequearemos el **tx hash** del mismo para guardarlo en los settings



# Acceder al contrato

- Obtendremos la address del contrato con el metodo “**GetContractAddress**” y guardaremos la direccion en los settings
- Probaremos llamar a “**CallContract**” para ver que nuestro contrato se implementó correctamente





# Cambiaremos el estado

- Le daremos nuevo saldo a nuestra cuenta
- Crearemos un nuevo Test que llame a la api “**callContractFunctionTxParameter**” (especial atención a los parámetros)
- Volveremos a llamar al método “**callContract**” del Test para ver si el saludo se actualizó correctamente



# Felicitaciones !

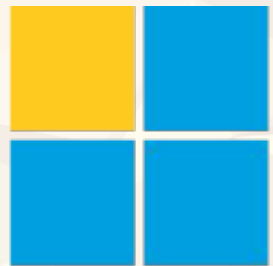
Has terminado el Workshop con éxito

# Sorteo !

Premio por haber participado



# ¡GRACIAS!



## .NET Conf CL v2018



NETConfCL



netconfcl



NETConfCL

## #EXPLOTA